A LOW-COST PARALLEL VLSI ARCHITECTURE FOR LOW-LEVEL VISION

Alberto Broggi[1] , Vincenzo D'Andrea[1] , Francesco Gregoretti[2]

[1] Dipartimento di Ingegneria dell'Informazione, Università di Parma - Italy
[2] Dipartimento di Elettronica, Politecnico di Torino - Italy

ABSTRACT

In this work a massively parallel VLSI architecture is discussed: PAPRICA, a mesh connected machine that can also simulate a pyramidal architecture. Its computational paradigm is based on the matching operator as defined in mathematical morphology. This description is equivalent to the Cellular Automata paradigm, which can simplify the design of PAPRICA applications in low-level vision. PAPRICA is currently simulated on a Unix workstation connected to a Connection Machine CM-2, while the hardware is under testing.

INTRODUCTION

Research in computer architectures for robotics has given birth to a wide variety of useful structures for parallel image processing. Mesh arrays, pyramids, hypercubes are only a few examples within the domain of massive parallelism. Even though none of these architectures is a panacea for solving the whole vision problem, the availability of processing power can make possible to solve subproblems in real-time. The architecture presented in this paper, PAPRICA, is an example of such an approach.

PAPRICA (PArallel PRocessor for Image Check and Analysis), is a massively parallel VLSI architecture [4]. It is a special purpose SIMD architecture, with a two-dimensional interconnecting mesh for interprocessors communication. One of the main goals in the development of PAPRICA is to keep the system powerful but simple enough to allow low-cost production.

A motivation for this approach is the use of PAPRICA in the framework of a pan-european project, PROMETHEUS, whose goal is is to develop "smart" sensors for assisting car drivers [1]. To this end, a Computer Vision system plays an important role, since most of the information available when driving has visual nature. A vision system, integrated into the sensor, should be able to provide the driver many different kinds of support, such as the detection of obstacles or the supervision in case of overtaking.

Beside the real-time requirement, the cost of such a smart sensor is an important point: it should be comparable to the cost of other car instruments. This argument leads to consider the use of PAPRICA in the PROMETHEUS framework for Image Processing. The architecture is indeed capable of performing, in real time, many low-level tasks relevant to visual analysis of traffic scenes.

In the next paragraphs after the presentation of the system architecture and its computational model, we will discuss PAPRICA simulator as well as some applications.

SYSTEM ARCHITECTURE

The PAPRICA system 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 (CU). The relationships between these units are shown in figure 1.

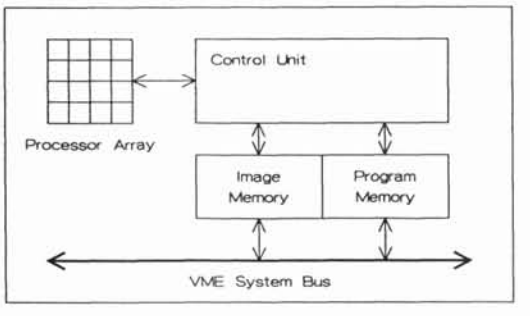

Figure 1: Block diagram of the PAPRICA system

The first prototype of the PA is composed of an array of 4×4 chips, 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.

The maximum configuration for PAPRICA system, will be achieved in the next version, and consists 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.

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, it is not possible to use the virtual processor approach implemented on the Connection Machine [9], 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 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. 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 value stored in the border processors of the PA is 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 correctly evaluate the previously invalidated results.

PAPRICA Image Memory can be rearranged run-time using the MEMORG instructions, which set the image height, width and deepness. 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 the image or to increase 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 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.

Performance analysis: The processing speed S_{pr} is a function of parameters that depend on the system technology and architecture (the number of Processing Elements 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`). The processing speed S_{pr} is defined as the number of pixel processed in the time unit for a given sequence of instructions or as the number of pixels over the total time needed for processing.

If the whole image to be processed fits into a PA of Q^2 elements, the time required for the processing will be the sum of three components. First, the time $Q^2 T_M$ required to load the array from the Image Memory whose access time is T_M; second, the time $L T_C$ required to execute L instructions, each with an elementary time T_C; and third, another time $Q^2 T_M$ to store back the results. S_{pr} will then be given by:

$$S_{pr} = \frac{Q^2}{2\ T_M\ Q^2 + L\ T_C} \tag{1}$$

Since T_C and T_M are of the same order of magnitude, if $L << 2Q^2$, S_{pr} reduces to:

$$S_{pr} \simeq \frac{Q^2}{2\ T_M\ Q^2} = \frac{1}{2\ T_M} \tag{2}$$

showing that the computational time is bounded by the I/O time. If, on the other hand, $L >> 2Q^2$, then equation (1) reduces to:

$$S_{pr} \simeq \frac{Q^2}{L\ T_C} \tag{3}$$

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 (4), obtained after few algebraic manipulations, shows the dependence of S_{pr} from the various known parameters; n_{upd} indicates the total number of `UPDATE` instructions into the program.

$$S_{pr} = \frac{(Q - \frac{2\ G}{n_{upd}})^2}{(2\ Q^2\ T_M\ n_{upd} + L\ T_C)} \tag{4}$$

Thus in a given program with L instructions, G of them being graphic operators which require full neighborhood access, it can be shown [2,3,4,5] that the maximum processing speed becomes:

$$(S_{pr})_{max} = \frac{4\ Q^2}{108\ Q\ G\ T_M + 9\ L\ T_C} \tag{5}$$

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 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 5 MegaPixel/s. Thus, the time required to process a 256×256 binary image with the architecture depicted above, in the general case is given by:

$$T = \frac{256 \cdot 256}{S_{pr}} \simeq \frac{65536}{70 \cdot 10^3} \simeq 0.9\ \text{s} \tag{6}$$

while the lower bound with $G = 0$ is $T = 0.013$ s.

COMPUTATIONAL PARADIGM

PAPRICA computational paradigm is based on the concept of *matching operator* $\oslash$, derived from the *hit-miss transform* described in [12]. This is a rather general approach which includes the other morphological operators as special cases [8]. Having defined the *N-dimensional frame* $\mathcal{F}_N \subset Z^N$ as a convex, size-limited, "rectangular" subset of the discrete Euclidean N-space Z^N, a *simple N-dimensional matching element* is a couple $\mathcal{Q} = (\mathcal{Q}_0, \mathcal{Q}_1)$, where both $\mathcal{Q}_0, \mathcal{Q}_1 \subset \mathcal{F}_N$ with the constraint that $\mathcal{Q}_0 \bigcap \mathcal{Q}_1 = \emptyset$. An *elementary matching* with a simple matching element is defined as:

$$A \oslash \mathcal{Q} \triangleq \{y \in \mathcal{F}_N \mid (y+b) \in A \quad \text{and} \quad (y+c) \notin A \quad \text{for every} \quad b \in \mathcal{Q}_1, c \in \mathcal{Q}_0\} \tag{7}$$

or, in terms of the erosion operator:

$$A \oslash \mathcal{Q} \triangleq (A \ominus \mathcal{Q}_1) \bigcap (A^c \ominus \mathcal{Q}_0) \tag{8}$$

A *complemented matching* $\oslash^c$ is also defined as:

$$A \oslash^c \mathcal{Q} \triangleq (A \oslash \mathcal{Q})^c = (A^c \oplus \check{\mathcal{Q}}_1) \bigcup (A \oplus \check{\mathcal{Q}}_0) \tag{9}$$

A *composite matching* with a *matching list* $\mathcal{Q}_\mathcal{L} = \{\mathcal{Q}^1, \ldots, \mathcal{Q}^k, \ldots,\}$ is the union of elementary matchings:

$$A \oslash \mathcal{Q}_\mathcal{L} \triangleq \bigcup_i (A \oslash \mathcal{Q}^i) \tag{10}$$

A simple 3×3 bidimensional matching element $\mathcal{Q}$ can be sketched using the following notation:  , where "x" is either 0, 1 or − if the corresponding pixel of the matching element is an element of $\mathcal{Q}_0$, $\mathcal{Q}_1$ or of none of the two. The center of the matrix is the pixel $(0,0)$ of $\mathcal{Q}$. A composite matching can be sketched as a list of simple matching operators:

$$\mathcal{Q}_\mathcal{L} = \left[\begin{array}{|c|c|c|}\hline x&x&x\\\hline x&x&x\\\hline x&x&x\\\hline\end{array}\right], \left[\begin{array}{|c|c|c|}\hline x&x&x\\\hline x&x&x\\\hline x&x&x\\\hline\end{array}\right]^c, \ldots, K\left[\begin{array}{|c|c|c|}\hline x&x&x\\\hline x&x&x\\\hline x&x&x\\\hline\end{array}\right], \ldots \tag{11}$$

where $K \in [2,4,8]$. The superscript ^{c} declares that a complementary matching $\oslash^c$ must be used in place of $\oslash$ for that specific element, while the numeric constant K is a short form for the list of K possible rotations of the elementary operator by $\frac{360}{K}$ degrees.

PAPRICA can perform matching operations using a fixed set of structuring elements, which form the *instruction set*. Some simple examples are given by the north translation operator (`NMOV`, expressed by the following structuring element: ) or by the vertical expansion operator (`VEXP`, expressed by the following list:  , 2 ), or by the border extraction operator (`BOR`, 8 ).

The instruction set can implement any structuring element using simple compositions following the mathematical morphology algebra [4].

A single Assembly instruction is formed by a concatenation of two orthogonal operations: a *graphic operator* (GOP) and a *logical operator* (LOP):

$$L_D = GOP(L_1)\,[LOP\ L_2]\,[\%A]$$

The former (GOP) operates on a first *source layer* (L_1) of an image (3×3 neighborhood), while the latter computes a diadic boolean function between the result of GOP and the central pixel of a second *source layer* (L_2). The result is stored on the *destination layer* (L_D). %A is an optional modifier which can be used to accumulate (ORing) the result of the operation on layer $L0$, which is used as an accumulator.

As an example, the synthesis of the matching element $\mathcal{Q} = \left[\begin{array}{|c|c|c|}\hline 1&-&0\\\hline 1&1&0\\\hline -&-&0\\\hline\end{array}\right]$ is depicted using mathematical morphology properties, reducing it to a sequence of simple matchings which can be easily implemented with PAPRICA instruction set and translated to PAPRICA Assembly language. The matching element $\mathcal{Q}$ can be decomposed in its two subsets $\mathcal{Q}_0$ and $\mathcal{Q}_1$, which can be decomposed again: $\mathcal{Q}_0$ can be rewritten as $\mathcal{Q}_0 = \mathcal{Q}_0^\alpha \oplus \mathcal{Q}_0^\beta$ where the two matching elements

$$\mathcal{Q}_0^\alpha = \left[\begin{array}{|c|c|c|}\hline -&-&-\\\hline -&-&1\\\hline -&-&-\\\hline\end{array}\right] \quad \text{and} \quad \mathcal{Q}_0^\beta = \left[\begin{array}{|c|c|c|}\hline -&1&-\\\hline -&1&-\\\hline -&1&-\\\hline\end{array}\right] \tag{12}$$

have been chosen since they belong to PAPRICA instruction set. The second part of the definition in (8) can be written as follows

$$A^c \ominus \mathcal{Q}_0 = A^c \ominus (\mathcal{Q}_0^\alpha \oplus \mathcal{Q}_0^\beta) \tag{13}$$

and for the chain rule property [8] (pag. 540) it can be rewritten as:

$$A^c \ominus \mathcal{Q}_0 = (A^c \ominus \mathcal{Q}_0^\alpha) \ominus \mathcal{Q}_0^\beta \tag{14}$$

Since in this case it is possible to substitute the erosion operator $\ominus$ with the matching operator $\oslash$, the previous becomes

$$A^c \ominus \mathcal{Q}_0 = (A^c \oslash \mathcal{Q}_0^\alpha) \oslash \mathcal{Q}_0^\beta \tag{15}$$

and can be easily expressed using PAPRICA assembly language notations as follows:

$$A^c \ominus \mathcal{Q}_0 = \texttt{VERS(WMOV(A}^c\texttt{))} \tag{16}$$

Using the (9), the previous expression can be coded in the following PAPRICA assembly program:

```
Lt = WMOV(L1) ~
L2 = VERS(Lt)
```

where `L1` is the layer containing the input binary image, `L2` is the output layer, and `Lt` is a temporary layer. `WMOV`, `VERS`, and the `~` symbol indicate respectively a west translation, a vertical erosion and the inversion of the result.

Using similar mathematical morphology properties, the first part of the definition in (8) can be rewritten as

$$A \ominus \mathcal{Q}_1 = \left(T \oslash \begin{array}{|c|c|c|}\hline - & - & - \\ \hline 1 & - & - \\ \hline - & - & - \\ \hline \end{array} \right) \bigcap \left(A \oslash \begin{array}{|c|c|c|}\hline - & - & - \\ \hline - & 1 & - \\ \hline - & - & - \\ \hline \end{array} \right) \quad (17)$$

where

$$T = \left(A \oslash \begin{array}{|c|c|c|}\hline - & 1 & - \\ \hline - & - & - \\ \hline - & - & - \\ \hline \end{array} \right) \bigcap \left(A \oslash \begin{array}{|c|c|c|}\hline - & - & - \\ \hline - & 1 & - \\ \hline - & - & - \\ \hline \end{array} \right) \quad (18)$$

The complete sequence of PAPRICA Assembly instructions is shown in the following:

```
Lt = SMOV(L1) & L1
L2 = EMOV(Lt) & L1
```

where `L1` is the layer containing the input binary image, `L2` is the output layer, and `Lt` is a temporary layer. `SMOV`, `EMOV` and the '`& L1`' symbols indicate respectively south and east translations, and a logical `AND` with the `L1` layer.

The final PAPRICA assembly code used to synthesize a matching operation with the matching element shown above is the following:

```
Lt = SMOV(L1) & L1
L2 = EMOV(Lt) & L1
Lt = WMOV(L1) ~
L2 = VERS(Lt) & L2
```

PAPRICA SIMULATION AND TUNING

PAPRICA programming environment gives the possibility of writing application programs using C language, with the help of two sets of functions:

- 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.

The low-level and hardware-dependent processing units are accessed by the high-level applications 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.

To overcome the speed limitation of the serial simulator, a parallel version was written for the Connection Machine CM-2. 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 two main reasons that led to the choice of the CM-2 as the hardware platform for the efficient 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 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.

An operator based on Kirsch [10] and Prewitt [11] operators was used as test for the performances of the simulator, 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 constraints 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.

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 CM-2 simulator 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.

PAPRICA APPLICATIONS

From the point of view of designing algorithms, it is more convenient to describe PAPRICA operations in terms of the Cellular Automata (CA) computational paradigm [5,6]. A CA [13] is a discrete dynamical system characterized by the following basic components:

- a regular network R of cells, whose connectivity defines the concept of topological neighborhood of a cell. A neighborhood of n cells $u \in R$ is designed by $I_{n,u}$.
- a finite set H, whose elements characterize the state $U \in H$ of each cell $u \in R$.
- a definition of the system's dynamic in terms of a local and uniform function of the values of the n cells of $I_{n,u}$, i.e. $F : H^n \rightarrow H$. The function updates the value of each cell $u \in R$ at time step t_i according to the values of the cells in the neighborhood of u at the previous time step t_{i-1}. Thus

$$\forall u \in R : U_{t_i} = F(V_{t_{i-1}} : v \in I_{n,u}) \qquad (19)$$

As it appears from the definition, the CA is a computational paradigm for massively parallel computation on a regular mesh, where each processor corresponds to a cell $u \in R$, the memory states are described by the elments of H, and the computational steps are the function F. Since PAPRICA is a massively parallel system it is possible to describe its operations in terms of CA. In fact, the mathematical morphology operations using size-limited structuring elements are equivalent to the description in terms of CA.

Despite the fact that each PAPRICA PE is directly connected only to its 8 neighbors, using only the direct links and boolean operations within a single PE, PAPRICA can synthesize via software any CA with arbitrary neighborhood (i.e. mathematical morphology operation with any structuring element).

According to equation (5) the maximum processing speed S_{pr} decreases with the increase of G (that is a function of the neighborhood dimensions). It is worthwhile to note that even with a very high neighborhood dimensions (about 60×60, corresponding to $G = 30$), a 256×256 binary image is processed in less than one second, as shown in equation (6).

CONCLUSIONS

For applications to low-level vision problems, the CA paradigm can map immediately all kernel-like filtering operations [5]. Examples of this class of operations are the enhancements based on the pixels gray values, gradient based filters, convolution operators, smoothing operators. Other operations require either the serialization of part of the algorithms (when there is the need of long-distance propagation of information), or the exploitation of higher level knowledge in the algorithm design.

PAPRICA simulator was successfully used for implementing several filtering algorithms [7]. Moreover it was also implemented an algorithm for extracting the road boundaries in a sequence of images taken by a moving car [5]. This algorithm makes use of a feature metric in order to identify the boundaries, taking into account domain specific knowledge, such as the position of the focus of expansion, measured by means of the optical flow.

References

[1] G. Adorni, "Artificial Intelligence for a safer and more efficient car driving", in *Proceedings SPIE*, Vol. 1007, Mobilre Robots III, pp. 310-318, Cambridge, 1988.

[2] 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.

[3] 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.

[4] 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.

[5] A.Broggi, V.D'Andrea, G.Destri, "Cellular Automata Machines as a Computational Paradigm for low-level Vision" in *International Journal of Modern Physics C*, Singapore, World Scientific, in press.

[6] A.Broggi, V.D'Andrea, "Elaborazione d'Immagini Mediante Automi Cellulari: un Algoritmo per l'Estrazione di Caratteristiche" in *I Sensi dell'Automa*, Trieste, Ed. LINT, in press., 1992.

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

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

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

[10] R.A.Kirsch, "Computer determination of the constituent structure of biological images" in *Computers and Biomedical research*, vol.4, n.3, pp.315-328, 1971.

[11]) J.M.S.Prewitt, "Object enhancement and extraction" in *Picture processing and psychopictorics*, B.S.Lipkin and A.Rosenfeld Ed., New York, Academic Press, 1970.

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

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