

The PAPRICA SIMD Array: Critical Reviews and Perspectives*

F.Gregoretto, C.Sansòè
Dip. di Elettronica
Politecnico di Torino, Italy

L.M.Reyneri
Dip. di Ingegneria dell'Informazione
Università di Pisa, Italy

A.Broggi, G.Conte
Dip. di Ingegneria dell'Informazione
Università di Parma, Italy

Abstract

PAPRICA project started in 1988 as an experimental VLSI architecture devoted to the efficient computation of data with two-dimensional structure.

The main goal of the project is to develop a subsystem that could operate as an attached processing unit to a standard workstation and in perspective as a specialized processing module in dedicated systems devoted to low level image analysis, cellular neural networks emulation, DRC algorithms. The architecture has been extensively used for basic low level image analysis tasks up to optical flow computation and feature tracking, showing encouraging performances even in the first prototype version.

The paper discusses the actual implementation and presents a critical analysis of the project, allowing to identify some crucial points of PAPRICA design (and of array processors in general) that must be carefully considered in the case of redesign.

1: Introduction

PAPRICA (PARallel PRocessor for Image Checking and Analysis) is a VLSI massively parallel SIMD architecture [10, 7, 4, 9], with a two-dimensional interconnecting mesh for interprocessors communication [12, 8, 14]. The main goal in its development was to keep the system powerful, but simple enough to allow low-cost implementation and able to follow the evolution of the technology.

The present state of the project consists in the evaluation of the first prototype of the architecture together with the critical analysis of the actual implementation in view of a possible redesign.

Section 2 discusses the present prototype; a deeper discussion can be found in [3, 7, 9]. Section 3 critically discusses the project with special attention to memory organization, I/O problems, and instruction set. Section 4 presents the possible evolution of the architecture, in order to overcome the present limitations. Section 5 ends the paper with some concluding remarks.

*This work was partially supported by CNR Progetto Finalizzato Trasporti under contract number 91.01031.PF93

2: The current implementation

The PAPRICA system operates as a specialized coprocessor attached to a general purpose host workstation and, as a whole, comprises 3 major functional parts, namely:

- A Processing Array (PA) hosting a 16×16 square matrix of 1-bit PEs each one with full 8-neighbors connectivity.
- A Program Memory storing up to 256k instructions, and a 3MB Image Memory with a 250 ns access time.
- A VLSI based Control Unit (CU) which drives the virtualization mechanism and controls the program flow.

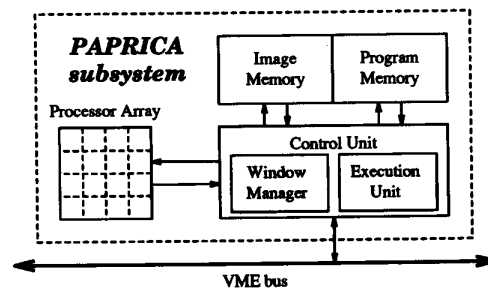


Figure 1: Block diagram of PAPRICA system

The relationships between these sub-units are shown in Figure 1. Figure 2 shows the photographs of the processing chip and of the complete system.

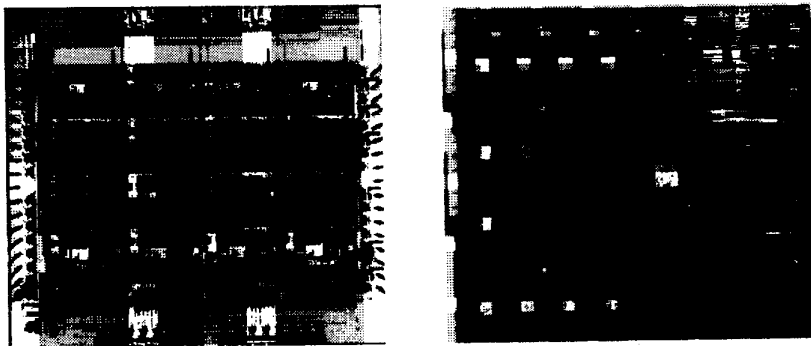


Figure 2: Photograph of the processing chip and of the complete board

The actual PA implementation consists of a two-dimensional mesh of single bit Processing Elements (PE) operating in a SIMD style and it is based on a set of custom VLSI circuits.

Each PE has a 64-bit internal register file; the corresponding bits belonging to different PEs form a binary layer L , which represents a window on the Image Memory. A single array instruction is formed by a concatenation of two orthogonal operations: a *graphic operator* (GOP) and a *logical operator* (LOP):

$$L_D = GOP(L_{S1}) [LOP L_{S2}] [\%A]$$

The former (GOP) operates on a first *source layer* (L_{S1}) of an image (using a 3×3 neighborhood). It can be selected within the following set: { INV, NMOV, SMOV, WMOV, EMOV, EXP, VEXP, HEXP, NEEEXP, ERS, VERS, HERS, NEERS, BOR, LS2}, which comprises single pixel translations, expansions, erosions, border extraction, and a specific operation dedicated to Design Rule Checking (DRC) of VLSI layouts.

The latter (LOP) computes a diadic boolean function between the result of GOP and the central pixel of a second *source layer* (L_{S2}). 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 L_0 , which is used as an accumulator. The internal structure of a PE is shown in Figure 3.

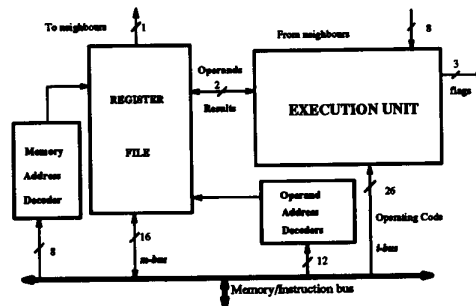


Figure 3: Internal structure of a Processing Element

The processors chip (hosting 16 PEs) has been designed using a full custom methodology and fabricated with a $1.5 \mu\text{m}$ CMOS technology, with a total complexity of approximately 35000 transistors on an area of 45 mm^2 . It is housed in an 84 pin PGA package, the measured yield is approximately 65% over a set of 120 samples and the average power consumption is approximately 0.3 W at 5V. The PE instruction cycle time is 500 ns.

The PA as a whole may operate in two different modes:

- **Memory mode.** It behaves like a standard RAM device composed of the 256 individual 64-bit PE registers for a total of 16k bits organized as $1\text{k} \times 16\text{-bit}$ words.
- **Processor mode.** All PEs execute the same array instruction fetched by the CU from the Program Memory.

Since the number of PEs is normally less than the number of image pixels used in generic low-level image processing applications, a significant part of the control unit has been dedicated to the implementation of virtual processors. PAPRICA serializes the computation in windows: the PA is loaded with a sub-window of the image, then the computation is

performed until a special control instruction (UPDATE) is reached, and finally the result is stored back into the image memory. 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 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 border processors of the PA are not valid any more. Thus, the next windows that will be transferred into the PA will be partially overlapped with the previous, in order to correctly evaluate the previously invalidated results.

Table 1 reports some performance figures of the actual implementation of PAPRICA, referring to basic low-level image processing algorithms performed on grey-level (8 bits/pixel) images. The values shown in Table 1 are:

- The number of clock cycles required to process a single sub-window loaded into the PA. It is a rough index of the amount of computation needed by the algorithm, and it is derived from the evaluation of the Assembly code.
- The neighborhood. It indicates the locality of the computation, and it is an index of the amount of I/O needed by the algorithm.
- The processing speed. It takes into account the overall performance of the system in the case of a 16×16 PE array.

Operations	Cycles	Neighborhood	Processing Speed
Addition, Subtraction	≈ 10	1×1	1.92 Mpixel/s
Maximum, Minimum	≈ 70	1×1	1.02 Mpixel/s
x and y Derivatives	≈ 80	3×3	1.16 Mpixel/s
Average	≈ 90	3×3	1.13 Mpixel/s
Kirsh Gradient	≈ 450	3×3	0.55 Mpixel/s
Prewitt Gradient	≈ 450	3×3	0.55 Mpixel/s
3×3 Gradient	≈ 490	3×3	0.52 Mpixel/s
Binary Thinning	≈ 50	17×17	0.25 Mpixel/s
Clustering (1 iteration)	≈ 2000	5×5	0.12 Mpixel/s
Optical Flow	≈ 845700	7×7	26 kpixel/s

Table 1: Performance figures of the actual implementation of PAPRICA

3: Critical analysis of the project

In the following sections the main problems observed in the present version of PAPRICA array are discussed in detail. The first one, specific to massively parallel architectures, refers to processor virtualization and to the physical allocation of the memory; the second one is related to the I/O limitation; finally, the external architecture of the PEs is discussed.

3.1: Memory organization and processor virtualization

The limitation in the amount of internal memory (64 bits) of each PE is the main bottleneck for the performances for two basic reasons.

- It is not possible to store internally a large amount of intermediate results, as often needed by computation intensive problems, thus requiring I/O operations to store the intermediate data externally.
- It is not possible to implement the well-known virtualization mechanism adopted by the Connection Machine CM-2 [18], where the computation is serialized within each processor (whose memory contains several data belonging to several pixels).

Whereas the first problem is a real limitation, the latter can be overcome using the virtualization mechanism described in section 2, which, on the other hand, offers some specific advantages. PAPRICA virtualization mechanism is less efficient than the one implemented on the CM-2 in terms of speed since it requires a lot of accesses to an external memory, but it offers the possibility to simulate a pyramidal structure [17] on the 2D mesh without any other hardware modification, and to implement a sort of *focus of attention* [19, 13].

- **Pyramidal emulation.** PAPRICA image memory can be rearranged run-time using special instructions, which set the image height, width and deepness. Moreover, it is possible to take advantage of the fact that for each parallel computation $2Q^2$ sequential accesses to the image memory are required (where Q is the linear dimension of the PA), fetching and storing back the data in a useful way. In fact, the data to be transferred into the PA may be not logically adjacent to each other in the image memory, 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.
- **Focus of attention.** An additional 64 bits register and simple logics allow each sub-window elaboration to be performed conditionally to one or more bits of the register. The implementation of this hardware extension does not involve major changes in the existing controller, since it is programmed on a XILINX module; currently it is under testing with the help of a system level software simulator, and the performances will be compared to the ones obtained with the existing hardware board. The efficient implementation of this sort of "multiple focus of attention" [6] is possible thanks to PAPRICA PE virtualization mechanism. The triggering idea is based on the consideration that sometimes, after the loading of an image sub-window into the PA, the elaboration may be useless, and thus the operation of storing back the results into the image memory would not be required. The choice whether to perform the elaboration or not depends on the characteristics of the complete set of data loaded into the PA internal registers.

3.2: I/O problems

The data transfer between the image memory and the processing array in the present version takes place on a common 16 bit data bus. The addressing mechanism is such that a

bus cycle selects a single PE to read or write a block of 16 bits belonging to the same pixel. This solution was chosen in the early stage of the design to simplify the architecture of the processing array chip, but it has proven to be somewhat inefficient. In fact, due to the windowing mechanism explained in section 3.1, in order to process a block of instructions its is always necessary to previously load the PEs with data from an image sub-window and to store them back after processing. Both data transfers have to be done on blocks of 16 image planes, while in many cases (almost all the practical ones) the number of necessary input and output planes is not a multiple of 16. This means that a significant amount of time is wasted transferring unnecessary data back and forth from the Image Memory.

The efficiency of the system can then be improved changing the addressing mechanism to allow an access by image plane instead of the present one: a bus cycle will then transfer 16 bits belonging to the same image plane from/to 16 different PEs. It has to be noted however that it is difficult to implement this transfer method with the present architecture: in fact, the sub-windows of the image are not multiple of 16 bits if graphic operators have to be processed. This means that, to maintain consistency, the controller has to shift and merge data from different locations during the load and store operations. The added delay would reduce consistently what is gained by optimizing the data organization.

3.3: Instruction set

PAPRICA computational paradigm is based on the concept of *matching operator*, derived from the *hit-miss transform* described in [16]. This is a rather general approach which includes the other morphological operators as special cases [11]. PAPRICA can perform matching operations using a fixed set of structuring elements, which form the *array instruction set*. This instruction set can implement any structuring element using simple compositions following the mathematical morphology algebra [3].

The actual version of PAPRICA has been conceived to run DRC algorithms and thus its instruction set follows a sort of CISC-oriented implementation, including also specific instructions needed to solve that task, such as binary border extraction (BOR), thickness verification of lines (LS2). The evaluation of the possible applications of PAPRICA [2, 5, 1] has led to the consideration that the instruction set should evolve toward a RISC implementation, allowing the execution of a minimum set of simple operations, such as translations of binary layers and logic operations between layers. Since the complexity of the instruction set determines the PE dimensions, the RISC choice allows the integration of a higher number of PEs on a single chip.

Furthermore, the possibility to enlarge the neighborhood involved in morphological operations has been investigated. With a relatively small PA, the execution of a large number of consecutive morphological operators requires a lot of I/O transfers (since there is a high overlapping between the sub-windows), decreasing the computational speed. With the increment of the PA dimensions, long sequences of graphic operators become feasible. On the other hand, the hardware complexity required to implement a larger neighborhood would increase the performances by a negligible amount. The RISC approach, implementing complex instructions as sequences of simple ones, just increase the length of the program and not its complexity.

4: Evolution and perspectives

Since the first prototype implementation, PAPRICA system has undergone a series of revision stages and a thorough investigation of a number of architectural modifications to overcome the previously described limitations. Basically the feasibility studies have outlined three main possible solutions which will be briefly described in this section.

- The first one is to maintain the same architecture of the original prototype and, taking advantage of the technological evolution and of the possible optimizations in the chip layout, to increase the size of the array. If the size becomes of the same order of magnitude of the images to be processed, the loading/unloading overhead due to the overlapping of windows required by the use of morphological operators would be reduced.
- The second one is to increase the addressing space of each elementary processor by the use of external memory elements. This choice logically simplifies the processor virtualization mechanism but limits the array size and presents additional implementation problems due to morphological operators.
- The third one is to change the physical interconnection topology of the processing elements from a two-dimensional mesh to a linear array moving from a *window* based to a *scanline* based processor virtualization.

4.1: Extension of the original design

The design of the prototype chip of the PAPRICA array has been carried out with a very conservative point of view, in term of technology, timing definition, layout of the circuit, and number of PEs in a dice. The original 4×4 PE chip occupied an area of $6.5 \text{ mm} \times 6 \text{ mm}$ using a 1.5 micron CMOS technology and recent redesign of the circuit in a slightly improved technology has given an area of approximately $5 \text{ mm} \times 5 \text{ mm}$. Therefore with an assessed 1.5 micron technology it is possible to integrate in a single chip of approximately $10 \text{ mm} \times 10 \text{ mm}$ an array of 8×8 PEs with an acceptable yield. Using a standard PCB technology at the system level an array of 4×4 chips would give an array of 32×32 PEs, still too small to overcome the performance limitations.

A feasibility study for 1 micron and 0.8 micron technologies has shown that in a single dice it would be possible to integrate an array of 16×16 PEs. In this case the problems arise at the system level since each chip would have a pin count of more than 200 pins and the power supply and speed requirements begin to depend strongly on the interconnection capacitances at the board level.

Advanced packaging and interconnection solutions have been investigated, such as Multi Chip Modules. A preliminary feasibility study has shown that with a thin film technology mounting bare chips directly bonded to the substrate it would be possible to integrate onto a $10 \text{ cm} \times 10 \text{ cm}$ substrate an array of 8×8 chip for a total of 16k PEs arranged into a 128×128 matrix, which is in the order of magnitude of the size of images for real applications. This being the limit given by today technology, an array of this size has two main drawbacks:

- The costs, related to the design of the new circuits, the fabrication, the testing of bare chips and the design and testing of the MCM modules, which have been estimated to be more than one order of magnitude of the current available prototype.
-

- The limitation in the addressing space of each PE, intrinsic to the architecture, cannot be easily overcome, since commercially available VLSI technologies do not offer the possibility of an efficient integration of large memories.

4.2: Extension of the addressing space

The limitation in the amount of internal memory could be overcome by the use of external memory circuits and providing a 1-bit channel from each processor to access it, as shown in Figure 4a. Since all PEs execute the same instruction, all memories are addressed at any time with the same address value, which may be generated by an external controller.

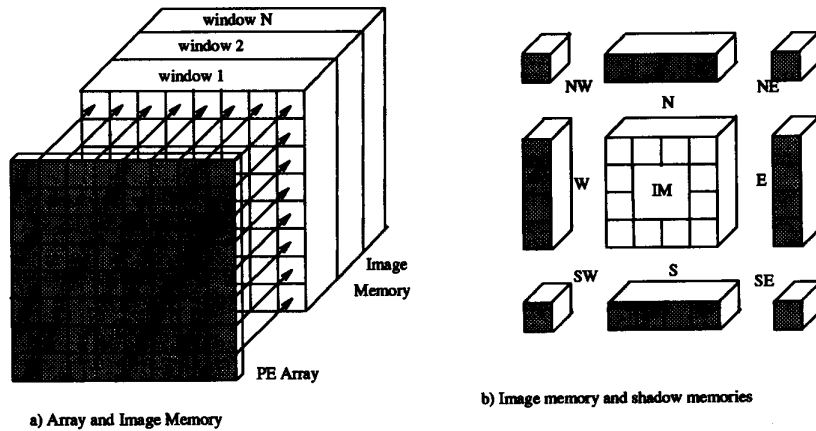


Figure 4: Architecture with external memory

This external extension of the registers has the main disadvantage of requiring a global communication channel towards the memory of the same parallelism as the size of the array; this fact limits the feasible size of the array itself and the parallelism achievable.

From the architectural point of view a large external memory provides an elegant alternative to the window scanning of the original PAPRICA prototype by the use of a processor virtualization scheme similar to that of the CM-2. The whole image to be processed may be partitioned into subimages which are stored in the external memory at different addresses and the scanning operation becomes just an address translation. Unfortunately, when dealing with morphological operators this solution presents the problem that when processing a given window, the neighboring data for the PEs on the border of the array are not available since they reside in the memory at a different address.

This can be solved, as shown in Figure 4b, by using additional memories, named *shadow memories*, which contain a copy of data stored in the border locations of the image memory. When the array processes a given subwindow, these auxiliary memories provide the border processors with input operands, but each one is accessed with a different address since data refer to different subwindows. When results are stored back, the shadow memories are written in parallel with the main image memory. This mechanism requires a very complex address handling which must be carried out by a dedicated circuit whose complexity is far

larger than that of the current window manager of the PAPRICA system. Moreover, since the data array resides in the external memory, the performances of the system are directly related to addressing and memory access times, and thus also the speed of the address generation mechanism becomes a critical point.

4.3: Architectural evolution

The problems encountered when trying to increase the performances of the system as exposed in the two previous sections are mainly due to the processor virtualization mechanism: it is always necessary to divide the image into sub-windows and, in order to execute morphological operators, it is necessary to know the contents of the pixels at the border of the sub-windows.

This problem is simplified if one of the image dimensions can be entirely loaded into the array, because in this case it is necessary only to scan the image in the other dimension. This is the basic reason underlying the idea to investigate a new architecture based on a linear array of PEs, big enough to process a complete image line in one iteration [15]. The linear architecture also better matches applications in real-time vision systems, where processing speed and data latency are the most critical factors. Such systems have several characteristics that have to be considered to optimize the low-level processor:

- data come serially from a sensor which generally scans the image by rows, so data become available one row at a time;
- image dimensions are fixed and are dictated by the resolution of the sensor;
- the results of low-level processing must be further elaborated by other stages managing higher level tasks.

In this environment, a one-dimensional array is the best solution to minimize data latency, because a row can be processed as soon as it is acquired from the sensor. Furthermore, the image width in such systems normally range from 128 to 1024 pixels, making perfectly possible to match the width of the array to that of the image.

The proposed architecture is depicted in Figure 5. The array chip will contain from 64 up to 128 PEs and the memory necessary to keep 64 binary layers of five image lines to handle the neighborhood (which can be extended to 12 pixels). An external large memory will contain the rest of the image, addressed by line and binary layer. To overcome the limits on the Image Memory-to-PA communication bandwidth, the width of the image memory data bus will be the same as the number of PEs. This allows the transfer of a complete line of image data in one memory cycle. This is feasible because, due to the linear organization, very few pins are needed for neighbors propagation between the different array chips, so it is possible to assign a considerable amount of pins to the Image Memory-to-PA communications. With this architecture, the operations that are necessary to process image line n are the following:

- the data contained in the internal array line memory are shifted in south-north direction by one position, while image line $n - 2$ is discarded;
 - input binary layers for line $n + 2$ are loaded from the external memory as south neighbors.
 - the program block is executed for line n ;
 - output binary layers for line n are saved to external memory.
-

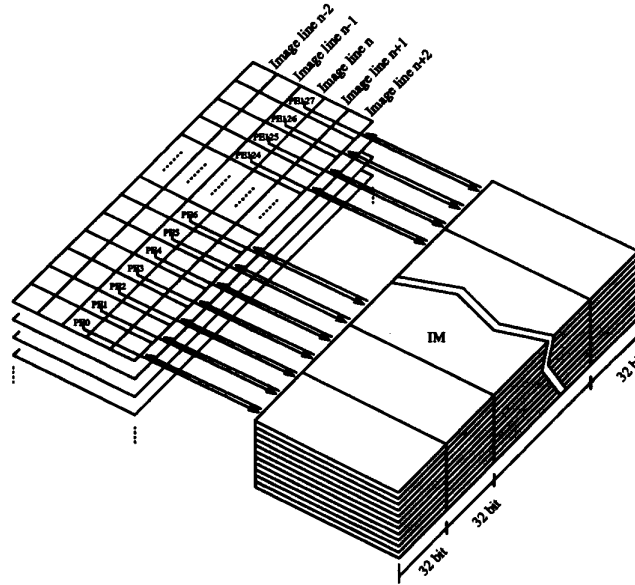


Figure 5: Linear array architecture

For real-time vision systems, it is possible to include in the array chip an input shift register to be able to acquire directly the image lines from the sensor, without having to store them previously into the Image Memory via external devices (host computer or dedicated hardware).

As for the other architectures, the same instruction block will have to be iterated for all the image lines. One possibility that is currently under evaluation is the insertion in the array chip of a program cache, to avoid reloading of the same code from external (relatively slow) memory as many times as the image lines.

In summary, the advantage of this architecture over the one depicted in the previous section is the great simplification of the memory organization (no shadow memories necessary) without performance degradation, at least when maximum sized images are processed: it should be noted in fact that, if it is necessary to deal with very small images, a lot of PEs will remain unused with a loss of performance compared with a square architecture with the same number of PEs.

4.4: Performance Analysis

From the point of view of performance the second and third solutions presented are basically equivalent, when considering an equal number of processors, since the difference lies in the interconnection topology. Moreover, with an external memory the performance is strictly related to its speed. Therefore in this section we will compare the large array solution with internal memory (see section 4.1) and the linear array (see section 4.3).

For the linear array the expected performance figures for a number of basic image pro-

cessing algorithms are reported in Table 2 assuming an array of 128 PEs with a 100 ns instruction cycle time, and a 100 ns read/write time access memory. The table reports separately the I/O figures and the basic algorithm timings. because a sequence of several operations can be executed on the same data.

I/O	Execution time
Read or Write - 1 bit matrix	$\approx 0.8\text{ns/pixel}$
Read or Write - 8 bit matrix	$\approx 7\text{ns/pixel}$
Algorithm	Execution time
Contouring of B/W image (1 instruction)	$\approx 0.8\text{ns/pixel}$
Skeleton of B/W image	$\approx 100 - 200\text{ns/pixel}$
Pattern matching (5×5)	$\approx 1.6\text{ns/pixel}$
Matrix summation (8 bit, integer)	$\approx 7\text{ns/pixel}$
3×3 pixel average (8 bit)	$\approx 30\text{ns/pixel}$
5×5 pixel average (8 bit)	$\approx 70\text{ns/pixel}$
Motion flow (max. ± 8 pixel per direction)	$\approx 0.5\mu\text{s/pixel}$

Table 2: Performances of a 128 PE linear array

If we extrapolate from data of Table 1 for the original array to a 16k PE implementation, assuming a speedup of 4 in the processor and memory cycle time, we get the data of Table 3 which shows the comparison between a 128 linear array and a 128×128 two-dimensional array without external memory. The figures in the table are obtained assuming an image of 128×128 pixels. The table clearly shows that the performance ratio between the two solutions has a maximum of one order of magnitude for purely arithmetic tasks and a lower value when morphological operations are required. Considering that the ratio of the number of PEs is 128:1, this rough evaluations, which do not take into account the complexity of the external memory addressing and control, show clearly the advantage of the solution with the external memory.

5: Conclusion

The paper presents a critical discussion of a project related to the design of a massively parallel architecture devoted to the analysis of two-dimensional data structures. The first prototype of the architecture is operational and its use on real applications shows its critical points. In view of the evolution of the system, the possible improvements have been discussed considering both new technological solutions and architectural redesign. The main

Algorithm	128 PE linear array	16k PE 2D array
Skeleton of B/W image	10 Mpixel/s	64 Mpixel/s
Matrix summation (8 bit, integer)	38 Mpixel/s	488 Mpixel/s
3×3 pixel average (8 bit)	24 Mpixel/s	289 Mpixel/s

Table 3: Comparison between a linear and a 2D array

alternatives are between a large PA with an internal register file and a square or linear structure with external memory capabilities. The solution with the external memory seems to offer the best performances to cost ratio. In this last case the linear array solution has a simpler memory organization and better behavior in the case of real-time applications (lower data latency).

References

- [1] G. Adorni, A. Broggi, G. Conte, V. D'Andrea, and C. Sansoè. High-level and low-level computer vision: towards an integrated approach. In F. E. Ardizzone, S. Gaglio, editor, *Trends in Artificial Intelligence*, pages 322-331. Lecture Notes in Artificial Intelligence, Springer-Verlag, 1991.
- [2] A. Broggi. Parallel and Local Feature Extraction: a Real-Time Approach to Street Boundary Detection. *IEEE Transactions on Image Processing*, 1993. Accepted for publication.
- [3] A. Broggi, G. Conte, F. Gregoretti, L. M. Reyneri, L. Rigazio, C. Sansoè, and C. Zamiri. PAPRICA. In *CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design*, pages 21-141. CNR - Progetto Finalizzato MADESS, Rome, 1990.
- [4] A. Broggi, V. D'Andrea, and F. Gregoretti. A low-cost parallel VLSI architecture for low-level vision. In *MVA'92 - IAPR Workshop on Machine Vision and Applications*, Tokyo, Japan, 1992. IAPR.
- [5] A. Broggi and A. Gandini. Parallel Image Clustering: A Real-Time Application for a Special-Purpose Architecture. In K. A. Hogda, B. Braathen, and K. Heia, editors, *Proceedings 8th SCIA*, volume 1, pages 297-304, Tromsø, N, May 25-28 1993. IAPR, NOBIM.
- [6] A. Broggi, S. Mora, and C. Sansoè. Enhancement of a 2D Array Processor for an Efficient Implementation of Visual Perception Tasks. In *Proceedings CAMP'93 - Computer Architectures for Machine Perception*, New Orleans, December 15-17 1993.
- [7] G. Conte, F. Gregoretti, L. M. Reyneri, and C. Sansoè. PAPRICA: a Parallel Architecture for VLSI CAD. In A. P. Ambler, P. Agrawal, and W. R. Moore, editors, *CAD Accelerators*, pages 177-189. North Holland, Amsterdam, 1991.
- [8] T. Fountain. *Processor Arrays: Architectures and applications*. Academic-Press, London, 1987.
- [9] F. Gregoretti, L. M. Reyneri, C. Sansoè, A. Broggi, and G. Conte. PAPRICA: an integrated approach to hierarchical morphology. Technical report, Dip. Ingegneria dell'Informazione, Università di Parma - Dip. Elettronica, Politecnico di Torino, 1993. Submitted for publication.
- [10] F. Gregoretti, L. M. Reyneri, C. Sansoè, and L. Rigazio. A chip set implementation of a parallel cellular architecture. *Microprocessing and Microprogramming*, 35:417-425, 1992.
- [11] R. Haralick, S. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. PAMI*, PAMI-9(4):532-550, July 1987.
- [12] NCR Corporation, Dayton, Ohio. *Geometric Arithmetic Parallel Processor*, 1984.
- [13] H. Neumann and H. Stiehl. Toward a computational architecture for monocular preattentive segmentation. In R. G. Hartmann and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*. Elsevier (North Holland), 1990.
- [14] S. Reddaway. DAP - A distributed array processor. In *1st Annual Symposium on Computer Architecture*, pages 61-65, Florida, 1973.
- [15] L. Schmitt and S. Wilson. The AIS-5000 Parallel Processor. *IEEE Trans. PAMI*, 10(3):320-330, May 1988.
- [16] J. Serra. *Image Analysis and Mathematical Morphology*. Academic-Press, London, first edition, 1982.
- [17] S. Tanimoto, T. Ligocki, and R. Ling. A prototype pyramid machine for hierarchical cellular logic. In *Parallel Computer Vision*. Academic Press, London, 1987.
- [18] Thinking Machines Corporation, Cambridge, Ma. *Connection Machine CM-200 Series - Technical Summary*, 1991.
- [19] J. M. Wolfe and K. R. Cave. Deploying visual attention: the guided model. In *AI and the eye*, pages 79-103. A. Blake and T. Troscianko, 1990.