4LP - LOW LEVEL LANGUAGE FOR THE LINE PROCESSOR " SYMPATI 2 "

Pascal ADAM - Didier JUVIN - Hassan ESSAFI
Pascal FERNANDEZ* - Jean-Luc BASILLE*

CEA SACLAY - D.LETI/DEIN/SIR 91191 GIF/YVETTE FRANCE
Tél :(1) 69.08.56.26

Laboratoire CERFIA - 50A Chemin des maraîchers 31077 TOULOUSE FRANCE
Tél : 61.52.13.52.

ABSTRACT
Image processing can take advantage of line processor structures. We have conceived a language that makes it possible to program a line processor in a way that eliminates many of the problems previously found when using the non-conventional structures.
We shall briefly remind the line processor concept and give some details about the structure our two laboratories are developing in a collaboration. Then we shall present the programming language facilities and illustrate these facilities with some examples. Finally some performances from simulated results will show processing time. This machine with up to 128 Processing Elements (P.E) has a theoritical power of 10 Mips per P.E.

RESUME
Les structures de processeur ligne sont particulièrement bien adaptées au traitement d'images. Aussi nous présentons dans ce papier un langage de bas niveau permettant de programmer le processeur ligne SYMPATI 2.
Ce langage élimine la plupart des problèmes de programmation rencontrés en général sur les calculateurs parallèles.
Nous rappellerons la structure de SYMPATI 2 calculateur SIMD développé conjointement par le CEA/D.LETI et le CERFIA Toulouse, avant d'introduire le langage 4LP et enfin nous présenterons les temps de calcul obtenus pour différents algorithmes.
Ce calculateur pouvant disposer de 16 à 128 processeurs élémentaires a une puissance théorique de 10 Mips par P.E.

1. THE LINE PROCESSOR STRUCTURE

Parallel structures may be considered as VON NEUMAN variations when duplicating some part of the so-called Von Neuman structure (BASI85). Among these different structures that might be involved in an image processing system, the line processor concept is a good choice particulary for pixel-level processing when considering the iconic domain, that is extracting features from the images (BASI86).

A line processor is a kind of processor array, a broader generic term that could include pyramids as well as the more traditional bidimensional processor arrays.
In accordance with the two fundamental criteria, efficiency and cost, we have good reason to believe that the line processor concept is a satisfactory middle road, particularly if we wish to design a image processor not only at a reasonable price, but also compatible with a personal computer configuration (BASI86).

A line processor exactly matches the linear structure as shown on figure 1. It also has certain characteristics that provide more possibilities than a simple linear structure would offer (BASI87).

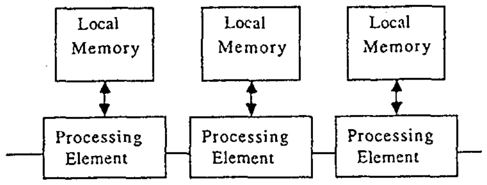


Figure 1: The linear structure

Let us present the main characteristics available in our system.

(i) . We have adopted the helicoïdal scheme for organizing the data in the processing element's memory.

According to the adressing system taking place in each processing element, it makes it possible to access, either by row or by column, to any N pixels segment, where N is the number of processing elements involved. This data organization preserves the topological properties of the image. So, two connected pixels of the 4 adjacency will belong to connected PE. This property permit the line processor to work either by row, either by column without any access conflict (two P.E will never require a value belonging to the same memory bank).

(ii) . The processor to processor connections makes it possible to access directly in one cycle to any point of the 3 x 3 neighborhod when a processing element processes a pixel of a row or a column segment. Furthermore it provides fast access to larger neighborhods up to 31 x 31.

Each processing element comprises two parts :

. addressing,

. processing.

The addressing system transforms the relative Euclidean coordinates of the row/column segment to be processed, into memory addresses for the considered pixels. So the data organization is transparent to the user. This addressing system also includes a masking module, which is very useful when only a window of the image is to be processed.

The processing part is more classical. It consists of a 16 bit wide ALU, a multiplier, a shifting module, a 16 bit registers Scratch-pad and an indicator set for the conditional sequences.

The different data paths make it possible for each processing element to access data in its neighbor's memory up to a distance 2 , or in the neighbor's scratch-pad, up to a distance of 3, right or left.

An independant data path is reserved for input/output operations.

2. DESCRIPTION OF THE PROGRAMMING ENVIRONMENT

One major difficulty for user with the non-conventional structure is the way to program it. Let us present now the language that we have conceived in order to eliminate such difficulties.

4LP is a low level language developed for the command unit with its own instruction set, some instructions are directly executable by the processing elements, others are executable by the command unit itself for the parallelism management.

Due to the SIMD structure 4LP is also a highly parallel language. Furthermore it makes it possible to focus one's attention on the job to be processed on one pixel. This job will be automatically processed for all the pixels of the considered window, in any of the scanning modes presented below (figure 2 - 3).

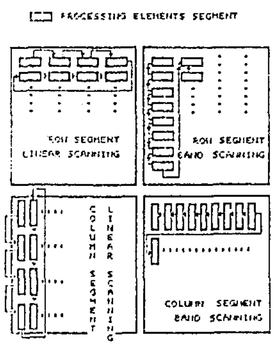
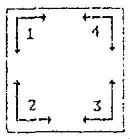

PROCESSING ELEMENTS SEGMENT
ROW SEGMENT LINEAR SCANNING
ROW SEGMENT BAND SCANNING
COLUMN SEGMENT
LINEAR SCANNING
COLUMN SEGMENT BAND SCANNING

FIGURE 2 : DIFFERENT SCANNING MODES



scanning mode direction

figure 3:

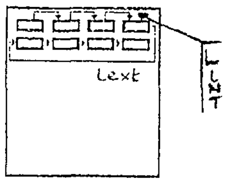


figure 4 :

3. PRESENTATION OF 4LP

<u>The basic program structure is the following one</u> :

```
PROG        Name; /* comments BLOC structure
 .                             . EXTERNAL LOOP
 Instructions;
 .                             .
 BLOC 1                        Instructions;
 .                             .
 Instructions;                 .
 .                             INTERNAL LOOP
 BLOC k                        .
 .                             .
 Instructions;                 Instructions;
 .                             .
STOP;                          LEND;
```

A 4LP program consists of PROG followed by a name and a semi-colon, lists of instructions, BLOCS, and a STOP directive followed by a semi-colon.

One BLOC consists of a LEXT (EXTERNAL LOOP) directive followed by an instruction list, a LINT (INTERNAL LOOP) directive an other instruction list and finally an LEND (END of LOOP) directive (figure 4).

LEXT, LINT are the command unit directives for the external and internal scanning loop start control, respectively.

LEND is the command unit directive for the end of LEXT and LINT loops (figure 4).

The number of blocs is unlimited. Initialization of the following parameters (images, windows, scanning, ...) can't be modified inside a micro-program.

THE DIFFERENT KINDS OF INSTRUCTIONS

- **ELEMENTARY INSTRUCTION**
- **NOP : NO OPERATION**
- **BRANCH INSTRUCTION : GOTO (COND) LABEL**
- **CONDITIONAL INSTRUCTION**

These instructions are all executed in a cycle of 120 ns.

THE ELEMENTARY INSTRUCTION

It consists of one destination, an ALU function, 0,1 or 2 operands, an optional indicator set up.

The functions are the usual logic and arithmetic ones, as well as the multiplication.

There are 4 binary indicators that may memorize one ALU flag.

<u>The principal destinations are the following ones</u> :

- A PE's personal scratch pad register.
  - S0.SP,...,S7.SP (8 bit registers)
  - D0.SP,...,D3.SP (16 bit registers)
  - X0.SP,X1.SP(16 bit)
- A PE's personal memory.
- A PE's special register for the addressing part.
- The ALU latch.

__The principal operands are following ones :__

- A scratch pad register belonging to either the PE, either one of its neighbors at a distance 1, 2, 3.
  . S0.SP -- PE's scratch-pad register.
  . S0.Li -- left neighbor PE, distance i = 1,2,3.
  . S0.Ri -- right neighbor PE, distance i = 1,2,3.
- A memory belonging to any PE, at a distance right or left up to 2.
- A constant value given by its address in the Command Unit memory.
- The ALU Latch

__Memory description__

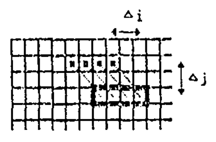
Each PE may only write in its own memory. Depending on the memory organization (helicoïdal or tabulated mode), the syntax is different.

1.) Tabulated mode :

Each PE has a special register named INDEX which contains the address of the memory value in a tabulated memory organization
EX : S0.SP = TAB _ the memory value at the address pointed by INDEX is Loaded in the S0.SP register.

2.) Helicoïdal mode) :

In this mode, the segment propagation is managed by the command unit (LEXT and LIND instructions). For each segment position, we access to any pixel by giving its relative coordinates i, i.



IMA.n° [Δi,Δj]

A PE can read any pixel value (except that i, j are limited to (+ 31). Due to the architecture, a direct memory transfer is possible between two PE if the distance is +1 or +2. This is very usefull because most of the time the PE asking for a pixel value is not the one which belongs it.

__Example of an elementary instruction :__

S0.SP, L = AND M1.0 (-3, +3), L I0 = C ;
S0.SP and the Latch (L) receive the result of the AND function between the LATCH value and the pixel (-3, 3) of the image number 0 window type 1. The indicator Io is set according to the Carry ALU flag.

This instruction is executed by all of the PE at the same time in 120 ns.

The BRANCH Instruction GOTO (cond) LABEL

(1) Branch when all PE's indicator selected are set up.
(2) Branch when any PE's indicator selected is set up.
(3) Unconditional branch.

__example__ :

```
A0 : S0.SP, L = MINUS M1.0(0,0),L   I1=GT;
GOTO A0 ALL1 ;
```

The first instruction (with A0 label) sets up I1 according to the GT flag of the result. The second one, tests the I1 indicator of each PE and if they are all set up, branches to A0 again.

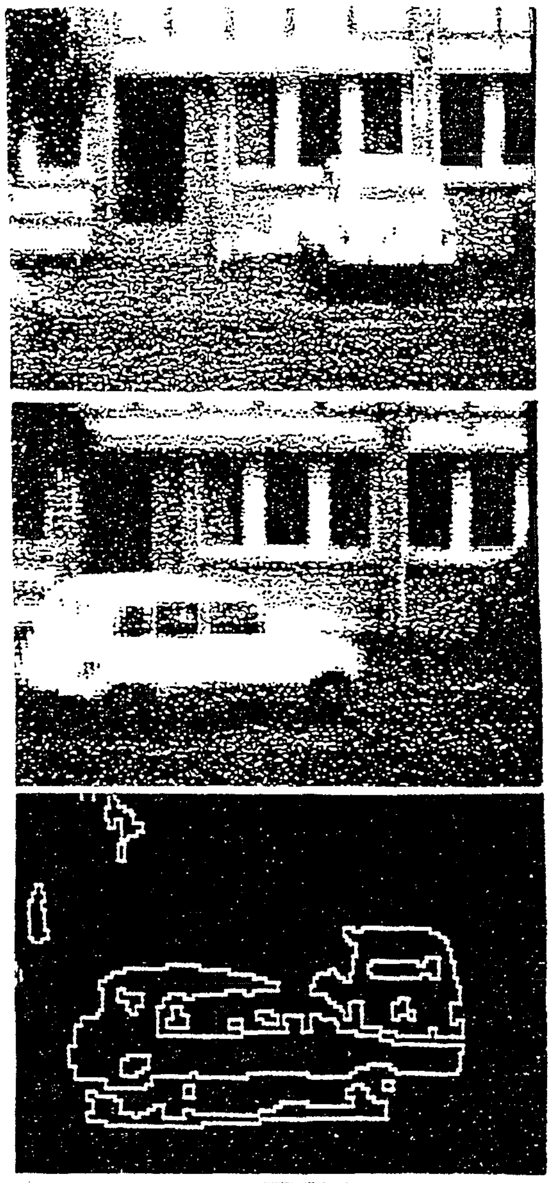
CONDITIONAL INSTRUCTION

S0.SP=ADD S0.SP, L /I0 ;
This instruction will be executed only by the PE which indicator I0 is set up, the other one will execute a NOP instruction.

PRESENTATION OF A SIMPLE MOTION EXTRACTOR

```
PROG MOTION;                 /* Motion extractor
/* Image 0, image 1 contain the same scene at
/* different times.
S0.SP = 16 ; threshold of binarisation.
/* scanning loops ............
LEXT;
    LINT;
/* pixel (0,0) of the 1st --> LATCH
   L        = M0.0(0,0);
/* difference between the pixel (0,0) of the
/* 2d. image and the latch, I0 is set up on the
/* sign of the result.
   L        = MINUS M0.4(0,0), L I0 = S;
/* in case of negative value, storage of inverse.
   L        = COMP L /I0;
/* Tresholding and storage in the image 1.
   L        = MINUS S0.SP,L I0 = S;
   M0.1(0,0)= MINUS1 /I0;
   M0.1(0,0)= NUL /NOT I0;
/* end of scanning loop.
LEND;
/* shrinked image (3x3 neighborhod)
LEXT;
    LINT;
/* AND (pixel value)
   L        = M0.1(-1,1);
   L        = AND M0.1 (0,0)          ,L;
   L        = AND M>0.1(-1,0)         ,L;
   L        = AND M>>0.1(-1,-1)       ,L;
   L        = AND M>0.1(0,-1)         ,L;
   L        = AND M0.1(1,-1)          ,L;
   L        = AND M<0.1(1,0)          ,L;
   L        = AND M<<0.1(1,1)         ,L;
   L        = AND M<0.1(0,1)          ,L;
/* storage of the AND result in image 2
   M0.2(0,0) = L ;
LEND;
/* Dilatation of the shrinked image
LEXT;
    LINT;
/* OR (pixel value) (3x3 neighborhod)
   L        = M0.2(-1,1);
   L        = OR M0.2(0,0)            ,L;
   L        = OR M>0.2(-1,0)          ,L;
```

```
        L       = OR M>>0.2(-1,-1)        ,L;
        L       = OR M>0.2(0,-1)          ,L;
        L       = OR M0.2(1,-1)           ,L;
        L       = OR M<0.2(1,0)           ,L;
        L       = OR M<<0.2(1,1)          ,L;
        L       = OR M<0.2(0,1)           ,L;
/* storage of the OR result in image 3
M0.3(0,0) = L;
LEND;
/* XOR between shrinked and dilated images to get /*
the edges of objects in motion.
LEXT;
    LINT;
        L       = M0.2(0,0);
        L       = XOR M0.3(0,0)  ,L;
        M0.5(0,0) = L;
LEND;
/* End of program
STOP;
```


Up, middle : images of the same scene.
Down : edges of objects in motion.

4. ALGORITHMS AND RESULTS

The prototype of the machine will be available by June 1989. We allready have the com plete development environment including the compi ler, desassembler, debugger and the simulator giving the exact computing time that will be required on the machine.

The algorithms, which results are presented now, have been simulated.

Results on 256x256x8 images.

Algorithm	32PE	128PE
Thresholding	1,2 ms	0,4 ms
Local variance	14,5 ms	3,6 ms
Convolution 3x3	5,3 ms	1,3 ms
Local minimum	4,5 ms	1,2 ms
Motion extractor	9,2 ms	2,2 ms
Reduction 256^2-> 128^2	7 ms	6,1 ms
Axial symetrie (horizontal axe)	10,8 ms	9,3 ms

5. CONCLUSION

We tried to show that 4LP is relevant for the pixel level processing. It provides an easy way for programming any algorithm operating on a window up to 31x31 without taking care of the data organisation.

The results obtained as for now lead to a very favorable efficient cost/ratio.

We reach the Quality factor of 6 for the Abingdon Cross Benchmark (Preston Kundall - CARNEGIE MELLON University) this is the best evaluated class for existing multi-processor machine.

We used YACC and LEX UNIX compiler development tools to write the 4LP compiler (kerni86).

6. REFERENCES

- (BASI85) BASILLE J.L., CASTAN S. - Multilevel architectures for image processing. 2nd Int. Tech. Symp. on Optical and Electro-Optical Applied Science AND Engineering. Cannes 2-6 décembre 1985.
- (BASI86) BASILLE J.L., DALLE P., CASTAN S. - Iconic and symbolic use of a line processor in Multilevel Structures. In : Duff MJB ed. Intermediate level image processing. London : Academic Press.
- (BASI87) BASILLE J.L., JUVIN D., ESSAFI H., LATIL J.Y. - Caractéristiques Architecturales d'un processeur ligne pour le traitement d'images, 6ème Congrès AFCET RFIA, Antibes 16-20 novembre 1987.
- (DUF81) DUFF M.J.B. and S. LEVIALDI - Languages and Architectures for Image Processing, ed. by authors, Academic Press, 1981.
- (KERNI86) KERNIGHAN B., PIKE R. - L'environnemnt de programmation UNIX, IIA ed. Inter Edition.