MVA'90 IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

MAPPING AN *N*N* IMAGE ON *P* PROCESSORS

Erwin R. Komen, Wouter B. Teeuw, Robert P.W. Duin and Pieter P. Jonker

Pattern Recognition Group,
Faculty of Applied Physics,
Delft University of Technology,
Lorentzweg 1, 2628 CJ Delft, The Netherlands.

ABSTRACT

An evaluation is made of hardware and software requirements to support window-, crinkle- and pyramid-mapping on a linear and a square processor array. The different mapping strategies also lead to different algorithms to solve image processing problems. Three image processing problems serve as an example of the algorithmic differences caused by the mapping methods. For the investigated 2D processor array, the pyramid mapping gives the fastest results.

1 INTRODUCTION

Image processing tasks can be calculated by using a number of Processing Elements (PEs) working in parallel. The question arises how an image can most efficiently be mapped on available PEs. In Section 2 different mapping methods are discussed. Implementation for the case of a CLIP4 processor array is described in Section 3. In Section 4 the mapping methods are compared for three image processing problems. The results of this comparison are discussed in Section 5.

2 THE MAPPING PROBLEM

2.1 Mapping theory

Solutions to the problem of mapping an $N*N$ image on P processors may result in different architectures: ways in which the P (PEs) are connected and cooperate.

Principally differing architectures are: the Linear Processor Array, the Square Processor array, the PipeLine, and the PyRamid (denoted as LPA, SPA, PL and PR respectively). Although the mapping problem leads to different architectures, the same PEs may be used (as argued in [6]). An LPA has P PEs configured in a line. Every PE will be able to have immediate access to 3-5 values: one of the values in its own memory (three for the AIS-5000; see [18]),one of its left, and one of its right neighbour.

An SPA is an array of PEs connected in a 2D grid in which each processor has access to the output values of its 'direct' neighbours. Each PE contains its own local memory to store the image pixel value corresponding to its position in the array. With a full-array, the image is as large as the SPA. Due to the large number of PEs needed for increasing image sizes (up to 4096*4096 for satellite images), this seems suited only for optical arrays such as the DOCIP [10]. Available arrays have sizes from 8*8 to 128*128.

The mapping types of image points on PEs that can be implemented on an LPA and SPA are the following:

- *Full:* For the LPA each of the P PEs processes one column. For the SPA each of the $\sqrt{P} \cdot \sqrt{P}$ PEs processes one pixel of the image.
- *Crinkle-wise:* For the LPA every PE is assigned to $\lceil N/P \rceil$ contiguous pixels of the row. The row is then processed in at least $\lceil N/P \rceil$ processing steps. This method is, for example, used by the PICAP3 [13]. For the SPA a sampled version of the image - with as many pixels as there are PEs - is mapped on the PEs of the PA. This method is used by the GRID [14].
- *Window-wise:* In an LPA (e.g. AIS-5000 [18]) the first P pixels of a row are processed with the P PEs, then the next P pixels etc. until the row is completely processed. This also takes at least $\lceil N/P \rceil$ processing steps. For an SPA(e.g. CLIP4 [5]) the image is divided in windows (scans) of the SPA size.
- *Helicoïdal:* This mapping - only for LPAs - is similar to window-wise mapping, but differs in the assignment of image points to PEs, that shift one point for every new row (see Figure 1). This mapping makes it possible to scan the array horizontally as well as vertically across the image. It is used in the SYMPATI-2 [11].
- *Pyramid:* This is an extension of crinkle mapping - but only for SPAs - in which images with less pixels than PEs are stored in a subset of neighbouring PEs. For example, a pyramid mapping on a 32*32 SPA has six levels containing arrays of sizes 1*1, 2*2, 4*4, 8*8, 16*16 and 32*32 stacked above each other. The base of the pyramid is a crinkle wise stored version of the entire image.

A processor mapping function (PMF) can be used to describe which image point is processed by which PE. For the LPA, a PMF shows in which PE at position *(x)* the image point *(i,j)* is stored. For the SPAs it shows in which PE at position *(x,y)* of the SPA the image point *(i,j)* is stored. The PMFs for most of the listed mapping methods are given in Table 1.

Table 1 Processor mapping functions

Name	LPA:	SPA:
Crinkle	$x = i/\lfloor N/P \rfloor$	$(x, y) = (i/k, j/k)$
Window	$x = i \bmod P$	$(x, y) = (i \bmod \sqrt{P}, j \bmod \sqrt{P})$
Helicoïdal	$x = (i+j) \bmod P$	
Full-size	$x = i$	$(x, y) = (i, j)$

with: $k = N/(\sqrt{P})$, $\sqrt{P}$ is an integer, and '/' integer div.

To demonstrate the different PMFs for the LPA and SPA, Figure 1 shows how an image of size 8*8 is stored in 4 PEs.

If an LPA uses crinkle or window mapping, this may involve extra overhead: for crinkle-wise mapping the neighbour values have to be obtained serially; for window-wise mapping the PEs which are on the array edge have to pass through their values from scan to scan (this can be done without overhead, as shown in the AIS-5000; [18]). LPAs which allow parallel or very fast access to their neighbours will be more suited for window mapping e.g. SYMPATI-2 [11].

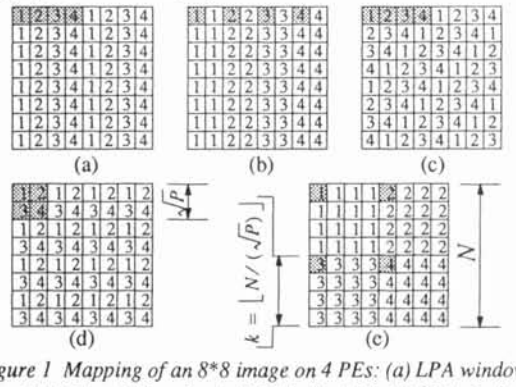

*Figure 1 Mapping of an 8*8 image on 4 PEs: (a) LPA window-wise, (b) LPA crinkle-wise, (c) LPA helicoïdal, (d) SPA window-wise and (e) SPA crinkle-wise.*

2.2 Window mapping

An SPA using window mapping divides the image in scans with a size of $\sqrt{P}*\sqrt{P}$ (see Figure 1d). Although every window can be processed individually, hardware or - if there are no such facilities - software should provide the values of the neighbours which are across the window borders.

Several methods exist to solve this 'edge-problem' [2][8]. In the experiments described in Section 3, the Edge Store Scanning (ESS) method is used. With ESS, every window is processed only after an edge around the PA is filled with the values of the pixels which are neighbouring the window (see Figure 2). Not many PAs are equipped with edge hardware to do this [8].

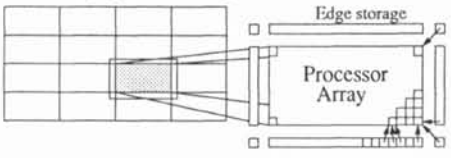

Figure 2 Edge store scanning.

The neighbourhood size used for window scanning operations willbe assumed - for convenience - not to be larger than the neighbourhood size available from the Processor Array which is used. There is scanning overhead of at least 2.5 to 4 due to the number of times a window is processed [12].

Window mapping is especially suited for SPAs which allow parallel neighbour access e.g. CLIP4 [5], BASE [15], GAPP [4].

2.3 Crinkle wise mapping

An SPA using crinkle mapping stores sampled versions of the image in the image memory. An example of crinkle mapping is shown in Figure 1e. The sampled versions are created using identical sampling frequencies. However, each time the sampling start positions differ.

In a crinkle wise stored image, neighbours in the memory are not neighbours in the image because sampled versions of the image are stored in the memory (see Figure 1e). Executing a neighbourhood operation on a crinkle wise stored image can not be done by using neighbourhood connections in an SPA.

The overhead with regard to processing a crinkle wise stored image depends on the number of neighbours that is used, because neighbour values have to be gathered one by one.

If arrays are used which allow parallel neighbour access, this facility is of hardly any use with crinkle mapping. Arrays which are designed for serial neighbour access are more natural to this mapping strategy e.g. MPP [1]. The GRID processor array is specifically designed for crinkle mapping [14].

2.4 Pyramid mapping

Pyramidal mapping is based on the pyramidal structure. A processing element in the interior of this pyramid has a local neighbourhood which consists of a father in the level above, eight neighbours at the same level, and four sons in the level below. This is shown in Figure 3. The original image is in the base of the pyramid. The other levels contain images derived from the base. A pyramid supports multi-resolution image analysis [3].

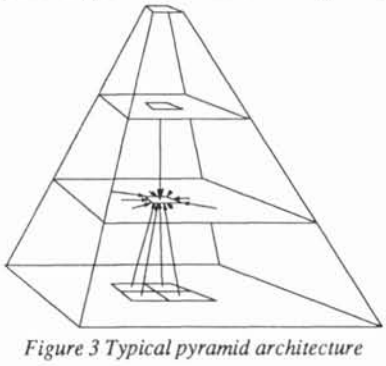

Figure 3 Typical pyramid architecture

With pyramidal *mapping*, a pyramidal data structure is mapped into the image memory of a small SPA. The upper levels of the pyramid have dimensions which are smaller than or equal to the dimensions of the SPA. Therefore they can be stored in (the upper left part of) a single memory plane in image memory. Processing them in the SPA is straight forward. The other levels are stored crinkle wise in image memory, in such a way that a father and his sons are stored in the memory of the same processing element. This allows a more natural up- and downwards processing in the pyramid. In the upper levels, shifts and masks have to be used to process up- and downwards in the pyramid.

The images which have to be processed with a pyramid are square in size. The dimensions have to be powers of two.

3 PROCESSOR MAPPINGS ON CLIP4

In this section we will describe how mapping methods can be implemented on a specific SPA: the CLIP (Cellular Logic Image Processor). This Processor Array was developed at the University College of London. For a full description of it we refer to [5]. The Delft CLIP4 has 64*32 PEs, 2048 additional memory planes, and does not posses hardware scanning facilities.

3.1 Window mapping on the CLIP4

A software scanning version of ESS has been programmed into the Delft CLIP4 using C4VM (CLIP4 Virtual Machine: a derivative of C; [7]). Programs which are written for the full-size image, can be run on the CLIP4 using ESS after recompilation.

While processing every window with the SPA, a special software edge storage is created by or-ring the pixels which are on the edges of the neighbouring windows (Figure 2a). These edges have to be transported from one side of the array to the other. This is done by a local neighbourhood operation followed by a global propagation operation.

3.2 Crinkle mapping on the CLIP4

Images which are 128*128 or 256*256 pixels in size have been crinkle wise stored in the Delft CLIP4 image memory. A function has been implemented to process these images.

An image of 128*128 pixels in size, for example, is stored in eight 64*32 memory planes. For reasons of symmetry, blocks of 2*2 (instead of 2*4) adjacent image pixels were stored in the memory of the same processing element. Therefore, 8 sampled versions of the image are actually stored (4 of the upper and 4 of the lower half). In fact the image is divided into two parts which are both crinkle wise stored.

The function for processing these images solves the problems which are described in Section 1.3. However, special attention has to be given to the artificial edge in the middle of the images, which is a result of the fact that the Delft CLIP4 processor array is not square.

The authors measured a worst case overhead (the average processing time per scan divided by the execution time for a single scan image) to be 28.8 (22.8) for processing a crinkle wise stored image of 128*128 (256*256) pixels [17].

3.3 Pyramid mapping on the CLIP4

A pyramidal data structure was implemented in the image memory of the Delft CLIP4 processor array. The pyramid consists of nine levels, the top level being 1*1, the base level being 256*256 pixels in size. The levels 0 (= top) through 5 (= 32*32 pixels) are stored in the upper left part of a 64 by 32 Delft CLIP4 memory plane. Level 6 (= 64*64 pixels) is stored window wise in two memory planes. The levels 7 (= 128*128 pixels) and 8 (= base) are stored crinkle wise. All the pixels of level 7, having the same father in level 6 are stored in the memory of the same PE. In total, level 7 occupies eight memory planes. The first four of these planes contain successively the upper left, upper right, lower left, and lower right sons of the pixels in the upper half of the image at level 6 of the pyramid. The last four planes of level 7 contain the sons of the pixels in the lower half of the level 6 image. The same relation as the one that exists between the levels 6

and 7 of the pyramid, holds for the levels 7 and 8 of the pyramid. The image of level 8 is stored in thirty-two memory planes. The pixels in the first four memory planes of level 8 represent the four sons of the pixels in the first memory plane in which level 7 is stored, and so on.

To store a binary pyramid in the Delft CLIP4's image memory, forty-eight memory planes are used. To store a grey value pyramid, eight times as many memory planes are needed.

4 ALGORITHMS

4.1 Edge detection in noisy binary images

The problem of detecting an edge in a noisy binary image is illustrated in Figure 4. Given the binary image as shown in Figure 4a, the original contour of the hand has to be recovered. The result of the algorithms is shown in Figure 4b.

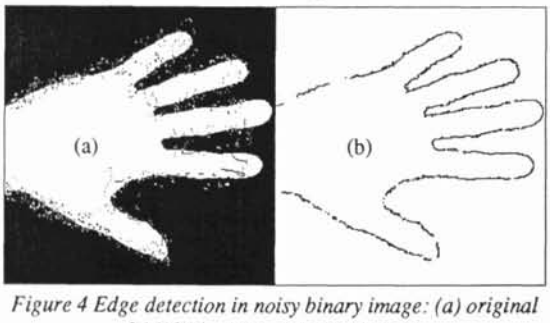

*Figure 4 Edge detection in noisy binary image: (a) original 256*256 image, (b) result image.*

The window and crinkle mapping version is as follows.

```
make three copies of the image
edge detection in the first copy
erosion, dilation, edge detection, and a dilation in the second
dilation, erosion, edge detection, and a dilation in the third
result is the logical AND of the three obtained images
```

The edge detection algorithm using pyramid mapping is as below [9]. Note that in this algorithm, the function *clean(...)* removes the noise pixels which is in- or outside the object (see Figure 4).

```
put image in base of pyramid-1
clean (pyramid-1)
put negation of image in base of pyramid-2
clean (pyramid-2)
dilate base of pyramid-2
result is the AND of the images in the bases of both pyramids
```

The pseudo code for the *clean(...)* subroutine is given below. For the results which appear in Table 2, the variable *higherlevel* was chosen equal to the number of the level which is just above the base. Changing *higherlevel* alters the size of the noise that is removed. The base of the pyramid has the highest number, the top (= level 0) has the lowest number.

```
for level = base-1 to higherlevel do
        pixels get value which is the AND of their four sons
od
edge detection in base and in higherlevel, result is saved
dilation higherlevel
for level = higherlevel+1 to base do
        pixels get value which is equal to their father value
od
erosion base image
logical AND of the base image with the saved edges
```

4.2 Finding the maximum grey value in an image

The problem of finding the maximum grey value in an image is illustrated in Figure 5. Given the grey value image as shown in Figure 5a, a binary image has to be constructed showing the positions of the image points with maximum grey value. The result of our algorithm is shown in Figure 5b.

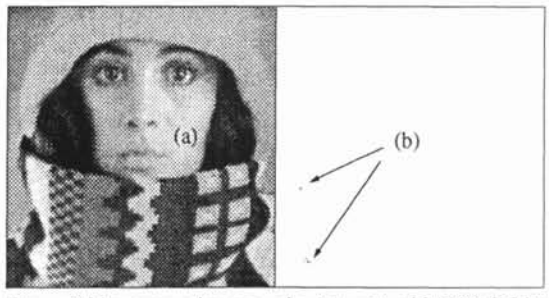

*Figure 5 Max grey value example: (a) original 256*256*8 bit image, (b) result image with positions of max greyvalue.*

An algorithm suited for a window or crinkle mapped image is described below. After execution, *maximum* contains the maximum grey value and the result-image shows its position.

```
all pixels in the result-image are set to one (1)
maximum = 0
factor = 2^7
stop = false
for all bitplanes of source image do
        most significant bitplane of source (not yet treated) is
                ANDed with result and result is put into help-image
        count = number of pixels set in the help image
        if count != 0 then
                result-image = help-image
                maximum = maximum + factor
        fi
        factor = factor / 2
od
```

The pyramid mapping version is as follows.

```
for level = base-1 to top do
        pixels get value which is maximum of their four sons
od
maximum = value in the top of the pyramid
for level = top+1 to base do
        pixels get value which is equal to their father value
od
EXOR all bitplanes of base with bitplanes of original image
invert all bitplanes base
result-image = logical AND of all the bitplanes in the base
```

In both algorithms, a test image completely filled with the maximum possible 8-bit number has been used.

4.3 Counting the fingers on a hand

An image of a hand has to be processed in such a way, that the fingers of the hand become separate objects. These are then to be counted. Given the binary image as shown in Figure 6a, the resulting finger images are shown in Figure 6b.

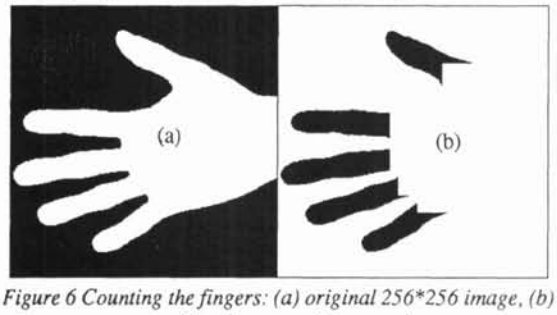

*Figure 6 Counting the fingers: (a) original 256*256 image, (b) resulting fingers to be counted.*

The window or crinkle mapping version:

```
erode the image 15 times {using all 9 points in a 3*3 nbhood}
dilate the image 16 times {same structuring element}
negation image
and image with the original image {now only fingers are left}
count the number of objects in the image
```

The pyramid mapping version is as follows [16].

```
for level = base-1 to top do
    pixels get value which is the AND of their four sons
od
find upperlevel of pyramid, which is not empty and which is as
        near to the top of the pyramid as possible
dilate upperlevel
for level = upperlevel+1 to base do
    pixels get value which is equal to their father value
od
negation of the base image
AND base image with original image {only fingers are left}
count the number of objects in the image
```

5 DISCUSSION OF RESULTS

The times which were measured on the Delft CLIP4 are given in Table 2. All timings were done with a minimum of system overhead, while performing the algorithm ten times (and dividing the result by ten).

Table 2 Performance of mapping methods on Delft CLIP4 (all times in ms.)

Mapping method:	edge detection:	Max grey value:	count fingers:
edge store scanning	438	55	1602
crinkle mapping	553	52	2165
pyramid mapping	423	3080 (52)	597

For the edge detection and the finger count algorithms, ESS is about 20% faster than crinkle mapping because ESS handles local neighbourhood operations more efficiently. The algorithm for the determination of the maximum grey value shows a difference between ESS and crinkle mapping caused by a slightly smaller overhead for ESS.

The maximum grey value algorithm performs very poorly for the pyramid mapping because multi-bit values are compared on the 1 bit PEs. Also, the simulation of a pyramid on the Delft CLIP4 has many layers (near the top) in the pyramid stored in a window-wise manner. Processing them up- and downwards causes a lot of overhead. As the pyramid mapping encompasses the crinkle-wise mapping, it is better to perform the window oriented algorithm in the base of the simulated pyramid (indicated by the value between brackets in Table 2).

With the pyramid edge detection algorithm, only the crinkle-wise stored layers of the pyramid are used. Openings and closings of a binary image in the window oriented algorithm are replaced by up and down processing in the pyramid algorithm. Therefore, the timings of the different algorithms are comparable.

The finger count algorithms fall into two parts: the isolation of the fingers, and the actual counting of them. The actual counting of the fingers takes the same amount of time for all three mapping strategies. The finger isolation is done using erosions and dilations for the window oriented algorithm, while using upwards ANDing and passing values through downwards for pyramid mapping. The latter involves less scans due to the smaller sizes of the pyramid levels.

As the algorithms for pyramid and window oriented mappings differ, the qualitative results can be different. The window oriented finger counting algorithm uses the prior knowledge of the size of the hand, whereas the pyramid-oriented algorithm is completely handsize independent. The pyramid oriented edge detection algorithm is a little bit sensitive to small shifts of the image, whereas the window oriented algorithm is not.

6 CONCLUSIONS

- *Window mapping:* If a processor array can access its neighbours in parallel and if scanning hardware is available, then one of the described window mapping techniques may be more suited.
- *Crinkle mapping* is good for operations which do not use many neighbourhood connections, e.g. in case of point operations (maximum grey value algorithm).
- *Pyramid mapping* is advantageous if the higher pyramid levels can be used for more global operations or operations that can be performed on sampled versions of the image (e.g. many erosions or thresholding a blurred image). Pyramid mapping is disadvantageous for comparing grey values in the top of the simulated pyramid. However, as pyramid mapping encompasses crinkle mapping, the crinkle algorithms can also be used in the base of the pyramid without reorganising the data.
- *Algorithms:* Different mapping strategies led to different approaches to solve image processing problems. If, for example, shrinking or expanding is used with window mapping, up- and downwards processing can be used in pyramidal mapping.

7 ACKNOWLEDGEMENTS

This work was supported by the Foundation for Computer Science in the Netherlands SION with financial support from the Netherlands organisation for Scientific Research (NWO) and the SPIN project Three Dimensional Image Analysis.

8 REFERENCES

[1] Batcher KE (1980) Design of a massively parallel processor. IEEE Transactions on Computers C-29:836-840

[2] Buurman J, Duin RPW (1988) Implementation and use of software scanning on a small CLIP4 processor array. In: Kittler J (ed) Lecture notes in computer science 301: Pattern Recognition, Springer-Verlag, London, pp269-277.

[3] Cantoni V, Levialdi S (1986) Pyramidal systems for computer vision. Springer-Verlag, Berlin Heidelberg.

[4] Davis R, Thomas D (Oct.1984) Systolic array chip matches the pace of high speed processing. Electronic design:207-218

[5] Duff MJB (1982) The CLIP4. In: Fu KS, Ichikawa T (eds) Special computer architectures for Pattern Recognition. CRC-Press, pp 65-86.

[6] Duin RPW, Jonker PP (1986) Processor arrays versus pipelines for cellular logic image operations. In: Young IT (eds)EUSIPCO II-I:Theories and Applications, Elsevier Science, pp1339-1342

[7] Fedorec AM, Otto GP (1988) The CLIP4 virtual machine reference. Report No. 88/1, Image processing Group, University College London.

[8] Fountain TJ, Postranecky M, Shaw GK (1987) The CLIP4S system. Pattern recognition letters 5:71-79.

[9] Gangolli AR, Tanimoto SL (1983) Two pyramid algorithms for edge detection in noisy binary images. Information processing letters 17:197-202

[10] Huang KS, Jenkins BK, Sawchuk AA (1989) Binary image algebra and optical cellular logic proc. design. CVGIP 45(3):295-345

[11] Juvin D, Basille JL, Essafi H, Latil JY (1988) Sympati 2, a 1.5D processor array for image application. In: EUSIPCO IV: Theories and applications, North-Holland, pp.311-314

[12] Komen ER (1990) Low-level image processing architectures compared for some non-linear recursive neighbourhood operations. Thesis. Delft Univ. of Techn., Pattern Recognition Group.

[13] Lindskog B (1988) PICAP3, An SIMD architecture for multi-dimensional signal processing; Dissertation No.176. Dept. of El. Eng. Linköping University, Sweden.

[14] Pass S (1985) The GRID parallel computer system. In: Kittler J, Duff MJB (eds) Image processing system architectures. Research Studies Press Ltd., Letchworth, pp 23-35.

[15] Reeves AP (Jan 1984) Parallel computer architectures for image processing. CVGIP 25:68-88

[16] Schaefer DH, Ho P, Boyd J, Vallejos C (1987) The GAM pyramid. In: Uhr L (ed.) Parallel computer vision. Academic Press, New York, pp. 15-42

[17] Teeuw WB (1989) Pyramid based image processing on a CLIP4 processor array. D-2 report, Pattern recognition Group, Faculty of Applied Physics, Delft University of Technology.

[18] Wilson SS (1988) One dimensional SIMD architectures - the AIS-5000. Multicomputer Vision, Academic Press, pp131-149