Clip 4 parallel processing system

T.J. Fountain, B.Sc., M. Phil., and V. Goetcherian, B.Sc., Ph.D.

Indexing terms: *Pattern recognition, Picture processing*

Abstract: A machine, Clip 4, is described, which is based on the ideas of parallel image processing developed at University College London. The major sections of the machine are outlined, including the processor array, which comprises 1152 custom-designed integrated circuits. A programming language has been developed and the application of the hardware/software system to a number of image-processing problems is shown. The processing speed of the system is indicated, and is typically several orders of magnitude better than moderately sized mainframe serial computers.

1 Overview

The Clip (cellular logic image processor) systems are based on the concept of parallel processing. The development of the series of Clip machines has been reported by Duff *et al.*[1] Under such a system, data, usually derived from a television camera picture, is presented to a two-dimensional array of interconnected processing elements, so that each processor is assigned to act upon the data from one point of the picture, and is permitted access to data from neighbouring points. Thus, any operation to be performed on the data occurs simultaneously at every point of the picture. This 'parallel' operation is the basis of the system's considerable speed advantage (typically 1 000 times faster) over conventional 'serial' processing techniques. Fig. 1 shows the Clip 4 system.

2 Hardware

Fig. 2 is a block diagram of the Clip 4 system. The major sections are as follows:

(*a*) The array of 9 216 processors. All the parallel data operations, on which the system depends, take place here.

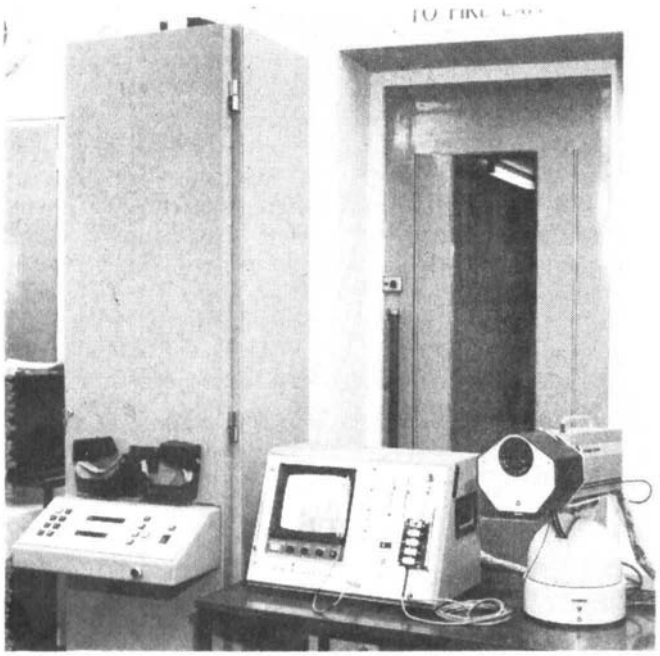


Fig. 1 *Clip 4*

Paper 917E, first received 25th March and in revised form 14th July 1980

Mr. Fountain is with the Department of Physics & Astronomy, University College London, Gower Street, London WC1E 6BT, and Mr. Goetcherian is with Logica Ltd., 68 Newman Street, London W1A 4SE, England

(*b*) The input/output structure, designed to interface data from the outside world to the array in an efficient manner.

(*c*) The control system, consisting of a local controller for program running and a minicomputer for program development.

(*d*) A data capture and display system consisting of t.v. camera and monitor console.

The central array consists of 96 x 96 basic processors, each connected to its eight neighbours. The sections of each processor (Fig. 3) operate as follows:

(i) Data input to and output from the array are achieved via storage bits A, which are connected from processor to processor to form shift registers.

(ii) Local storage of data during processing occurs in D, a 32-bit r.a.m. This local storage of data during processing is essential for the high operating speed of the array.

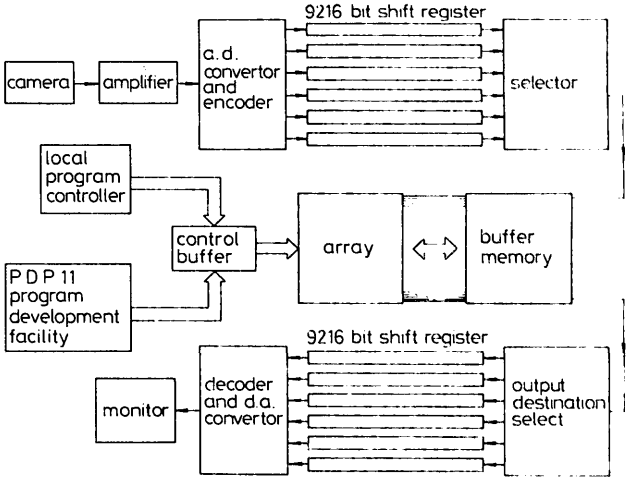


Fig. 2 *Clip 4 system block diagram*

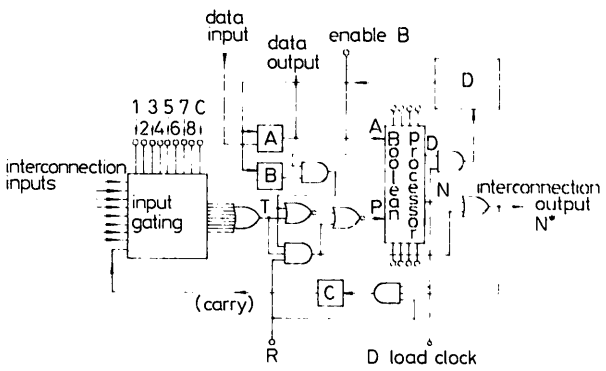


Fig. 3 *Clip 4 processor circuit*

0143–7062/80/050219 + 06 $01·50/0

(iii) Inputs from neighbouring processors are gated to produce a composite function T.

(iv) Two inputs to the central processor logic are derived as follows:

'A' is always data drawn either from the D registers or from an external source via the 'A' shift registers. The 'P' input is derived from the D register, from the composite interconnection input T, or from arithmetic operations in the array (described below).

(v) The processor produces any Boolean combination of A and P on two output channels, D and N. The D channel is stored in the D register. The N function, which need not be the same as the D output, is passed as an output to neighbouring processors.

(vi) By means of an additional control line and the storage bit C, the array can perform arithmetic operations in two modes.

Connectivity between processors in the array is determined by the required picture tessellation. The Clip 4 machine allows either square or hexagonal tessellation, under software control (Fig. 4).

In either array mode, each cell has a gated connection to each adjoining cell, the gating on these lines providing for the implementation of directional algorithms within the array.

Inputs to the array are derived from the central area of a t.v. scan. Since the Clip 4 processors are made up of binary circuit elements, the analogue signals derived from the t.v. camera undergo a process of 'grey scale encoding'. Each input data point is stored as a six-bit binary word in the input store.

Since data is input to the Clip 4 array via a single memory bit per cell, a number of different modes are provided for selecting partial data sets from the full set of grey scale information available in the input stores. These are:

(*a*) *Threshold cut:* A six-bit binary number is set in the appropriate control instruction. Any data point where the stored six-bit binary value is greater than the number in the instruction is passed to the array as a '1'.

(*b*) *Inverse threshold cut:* Here any point whose value is less than the number in the instruction is passed to the array as '1'.

(*c*) *Level select:* Only those data points having the same value as the number in the instruction are passed to the array as '1's.

(*d*) *Bit select:* In order to facilitate rapid transfer of the entire grey scale picture into array storage, it is possible to select and transfer each of the bit strings individually. Thus in six operations (one for each of the six bits) the entire picture information can be entered into the array stores. Output pictures are displayed continuously on a monitor screen, except when the output stores are being updated with new data from the array. The system permits manual selection of any combination of the output stores, and various modes of mixing the output display with the picture direct from the t.v. camera or input stores.

A channel is provided in the I/O whereby data may be transferred between the array and, for example, the core store of an interfaced minicomputer.

Additionally, compressed data from the array, for example a count of the number of objects in a picture, can be transferred to a control register for analysis purposes. The contents of this register are also presented for inspection on the console.

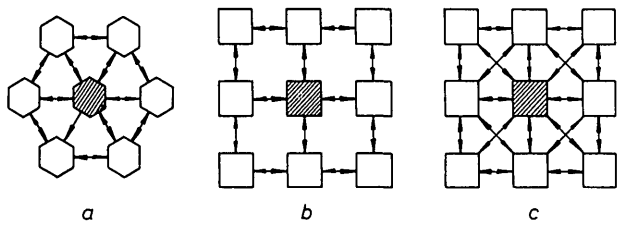


Fig. 4 *Possible Clip 4 tessellations*

a Hexagonal
b Square – 4
c Square – 8

3 Software

3.1 Programming language

Since the Clip structure is unlike that of a conventional computer, a special programming language has been developed for handling operations within the array of processors. The array instructions are as follows:

(i) *Load:* These load either the A or B register bits in each processor from a specified address in the associated D register, e.g.

Load A from D_{18} : LDA 18

Load B from D address specified, plus the contents of Register 3 :* LDB 10 (3).

(ii) *Set:* This sets up all required control bits for array operation but does not start the operation. It has the following operand fields:

(*a*) that specifying the processor output in terms of the A and P inputs

(*b*) that specifying the connectivity directions to be activated, and the propagation output

(*c*) a composite field specifying the array configuration, the array edge state, and the arithmetic operation control functions.

(iii) *Process and store:* This initiates a process set up by the previous SET instruction and stores the result in a specified D address, e.g. PST 18. Thus a typical one operation process has the form:

LDA 18
LDB 20
SET (the required operators)
PST 24

In addition to these 'array' instructions there are input/output instructions, conditional and unconditional branch instructions, and instructions to permit certain register operations.

3.2 Typical parallel operations

(*a*) *Binary:* Typical single-instruction operations on binary data are: noise removal†, removal of edge-connected objects, shrink by one layer†, expand by one layer†, edge finding, shift†, determine coincidence of two images†, and label (using one image to mark part of a second). Detailed descriptions of these and other array operations are given by Duff *et al.*[1]

(*b*) *Grey scale processing:* In order to implement, for example, a Forsen gradient on a grey level picture, Clip requires about four local operations per bit plane. For a six-bit picture this would take about 650 μs.

*One of 16 indexing registers in the controller

†Each of these local operations takes 25 μs to complete. The time taken for the remaining operations is dependent on the amount of data propagation involved

It is estimated that these processing times, which are array-size invariant for a parallel processor but not for a serial computer, and which refer to the Clip 4 array size of 96 x 96 points, would be about 1 000 times longer on a serial machine such as an IBM 360.

(*c*) *Arithmetic:* Two basic types of arithmetic can be implemented in the Clip array: binary column arithmetic and bit plane arithmetic. In the first type of operation, a maximum of 96 operations can be performed simultaneously and the maximum result size is a 96-bit number for each operation. As an example, the simultaneous multiplication of 96 pairs of 16-bit numbers takes about 2 ms, giving an effective multiplication time per pair of less than 25 μs. In bit plane arithmetic, 9 216 operations are performed simultaneously and the maximum result size is 16 bits. The overall time taken to add 9 216 pairs of 16 bit numbers is about 450 μs, giving an effective addition time per pair of 50 ns.

4 Applications

The two-dimensional binary array described in Section 2 forms the basis of all the image processing performed using Clip 4. It is true to say that, at the very low level, Clip can handle only binary data. This does not mean, however, that we are confined to binary imagery. In this Section we will present some applications of Clip to image processing, giving examples of binary and grey image algorithms.

4.1 Bit stacks and grey images

To represent grey images in an array form, we use several D storage planes per image, as shown in Fig. 5.

D_0 contains the least significant bit of the image and D_n the most significant bit. To represent an 8 grey tone image, for example, we need 3 bit planes and, in general, for an N grey tone image we need $n = \log_2 N$ bit planes. Given two bit stacks of sizes m and n planes, respectively, one can add them together, subtract one from the other or multiply them together using software techniques. In addition to this, one can perform 'within-stack' operations, i.e. the value of a picture element can be added or subtracted from the value of any of its neighbouring pixels. These operations are, of course, simultaneous over all picture elements, and hence the time taken is independent of the array size.

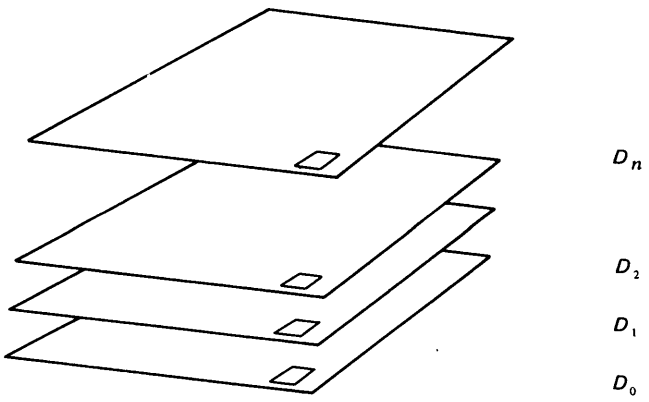


Fig. 5 *D-level representation of grey images*
D_n Most significant
.
.
.
D_2
D_1
D_0 Least significant

4.2 Simple edge detection

Fig. 6 shows an example of a simple edge-detection algorithm applied both to binary and to grey images. The original image (Fig. 6*a*) is a Landsat photograph of N.W. Wales, and Fig. 6*b* is a binary thresholded version of the data. The statement of the parallel algorithm, in words, is:

> 'Consider each picture point (simultaneously). Subtract from the value of that picture point the minimum value in its immediate surround.'

By the immediate surround we mean the 3 x 3 window centred on the picture point. The binary edge detection (Fig. 6*c*) can be performed in one instruction cycle of Clip, i.e. in ~ 25 μs. The grey algorithm takes a considerably longer time than this to run (Fig. 6*d*).

4.3 Labelling

The labelling operation consists of marking an object of interest in the image field and isolating it from all other objects in the image. The 'label' (or 'pointer') can consist of one single point, a set of points or a second object in another image. Labelling involves propagating a signal from the 'pointer' to all the '1' cells (black) that are connected to that point in the image.

An example of binary labelling is shown in Fig. 7. The image shows a cross-section of rat testes cells viewed under a microscope (Fig. 7*a*). The 'pointer' is seen in Fig. 7*b* and the isolated central cell in Fig. 7*c*.

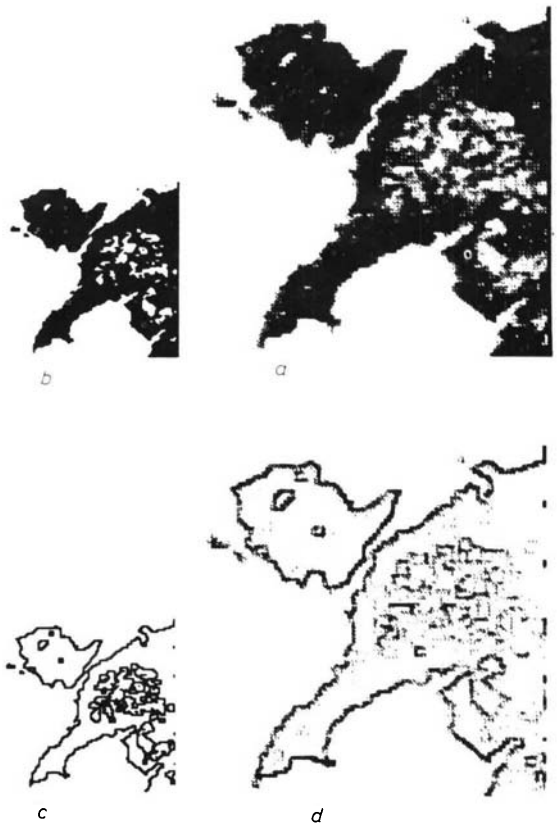


Fig. 6 *Edge detection algorithm*
a Original
b Thresholded
c Binary edge detection
d Grey level edge detection

Fig. 7 *Binary labelling*

a Original
b Pointer
c Isolated cell

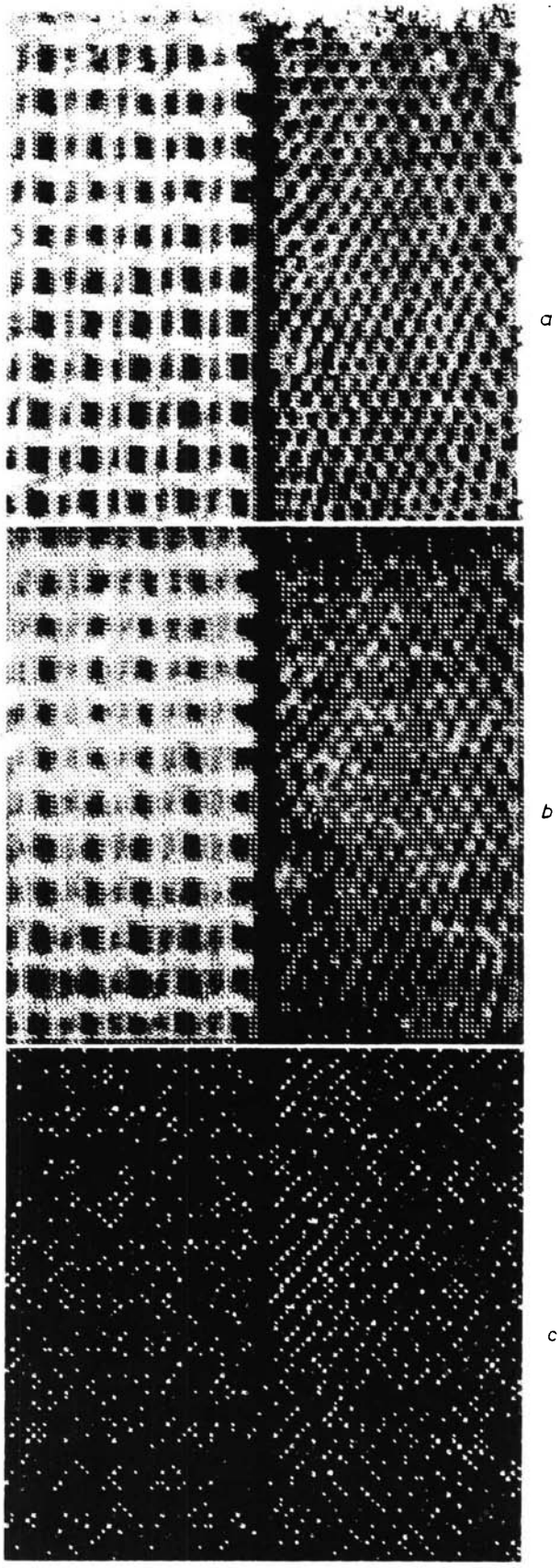


Fig. 8 *Image filtering*

a Original
b Binomial filter
c Laplacian filter

Table 1: Horizontal neighbour addition

a_1 a_2 a_3 a_4	a_1+a_2 a_2+a_3 a_3+a_4	$a_1+2a_2+a_3$ $a_2+2a_3+a_4$ $a_3+2a_4+a_5$
b_1 b_2 b_3 b_4	b_1+b_2 b_2+b_3 b_3+b_4	$b_1+2b_2+b_3$ $b_2+2b_3+b_4$ $b_3+2b_4+b_5$
c_1 c_2 c_3 c_4	c_1+c_2 c_2+c_3 c_3+c_4	$c_1+2c_2+c_3$ $c_2+2c_3+c_4$ $c_3+2c_4+c_5$
Original	One operation	Two operations

4.4 *Two-dimensional operations*

The within-stack arithmetic operations mentioned previously can be used to perform two-dimensional filtering operations on images. To see how this is achieved, let us consider what happens if we repeat the process of, say, nearest neighbour addition several times.

Considering the horizontal direction first, Table 1 shows several steps in the addition of nearest neighbour element values. If we denote the nearest neighbour addition in the horizontal direction by $\Sigma_x(\Pi)$, where Π is the grey image on which the operator Σ_x is acting, this is defined at the local level by

$$g = g_{xy} + g_{x+1,y}$$

where g_{xy} is the value at co-ordinate (x, y).

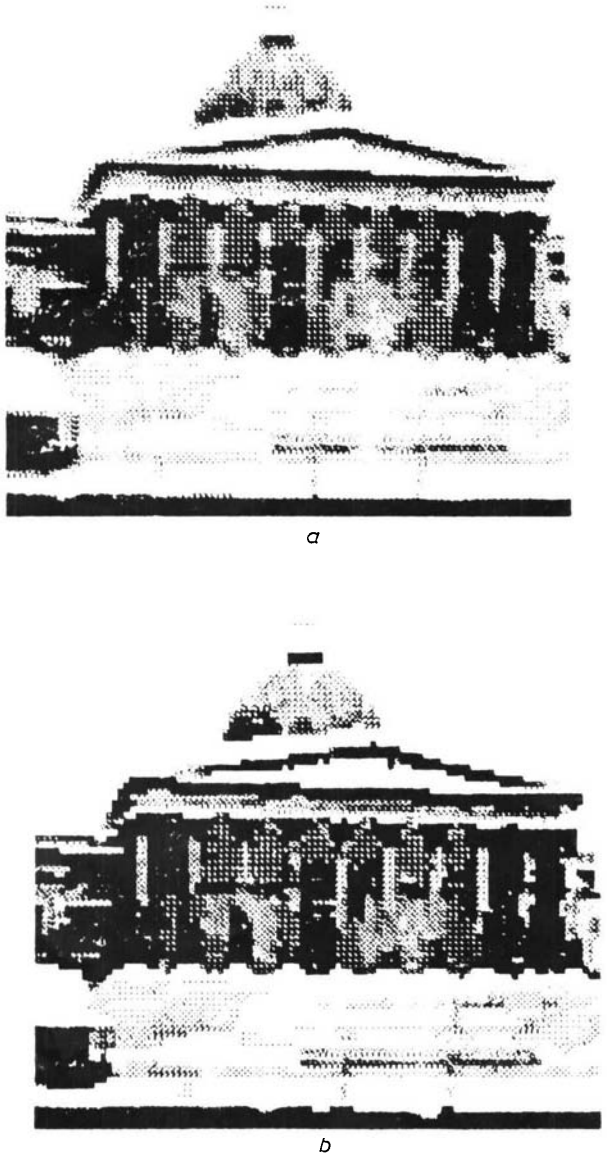


Fig. 9 *Image enhancement*

a original
b enhanced image

Similarly, we denote the nearest neighbour (horizontal) difference operation by $\nabla_x(\Pi)$, where:

$$g_{xy} = g_{xy} - g_{x+1,y}$$

It is evident, from Table 1, that the weights of the neighbouring cell values that go to make up the new value of an element follow the binomial sequence, i.e. $(\Sigma_x)^2(\Pi)$ has the coefficients (1, 2, 1), $(\nabla_x)^3$ (Π) has coefficients $(1, -3, 3, -1)$, etc. If, in extending the one-dimensional case above, we take into account a second direction (vertical), we end up with a two-dimensional 'binomial' filter e.g.

$$(\Sigma_x)^2(\Sigma_y)^2(\Pi)$$

The effect of this filter on the image is the same as that of convolving the original with a window of weights.

1	2	1
2	4	2
1	2	1

It produces a low pass effect by 'smoothing' local details.

Fig. 10 *Skeletonisation*

a Original
b Thresholded original
c Binary skeleton
d Grey level skeleton

Similarly, $(\nabla_x)^2$ $(\nabla_y)^2$ (Π) has the same effect as convolving the image with:

1	−2	1
−2	4	−2
1	−2	1

This is the well known Laplacian operator in digital images. It acts as a spot detector. Examples of the two filters above applied to the data in Fig. 8*a* are given in Fig. 8*b* (binomial) and 8*c* (Laplacian). The ideal of binomial filters has been more thoroughly investigated by M. James.[2]

4.5 *Image enhancement*

The aim of image enhancement is to sharpen the transition between 'background' and 'object'. This does not add any new information to the image (in fact some information may be lost in the process), but it makes images easier to interpret.

The parallel algorithm used to perform the enhancement shown in Fig. 9, from the point of view of each picture point, is as follows:

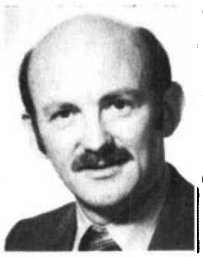
'Look at the maximum and the minimum values in your 3 x 3 surround. Which is closer to your own value? Alter your value to whichever of the two is closer.'

This process can be iterated over many cycles and can be shown to settle to a steady-state (nontrivial) result.

4.6 *Skeletonisation*

Skeletonisation is used in image processing to reduce an object to a minimal form which still retains the topological information of the original image (connectivity, holes, Euler number, etc.)

The binary skeletonisation algorithm was devised by Cordella *et al.*[3] The algorithm was extended to cover grey images by one of the authors and a detailed description of the two algorithms is given in reference 4. Examples of these algorithms operating on human chromosomes are shown in Fig. 10.



T.J. Fountain graduated from Queen Mary College, London, with an honours degree in physics, in 1966. After two years in industry studying the properties of thin-film semiconductors, he joined the Image Processing Group at University College London. Having presented his Master's thesis on low temperature radiation detectors in 1972, he began work on the Clip series of image processing machines and is now the Group's senior engineer, with overall responsibility for the production of the Clip 4 system.

5 Summary

A machine, Clip 4, based upon the idea of parallel image processing, is being developed at University College London. The array of processors at the core of the machine is implemented with custom l.s.i. circuits, and operates on binary data.

A special programming language has been designed to handle topological operations, and software techniques have been developed to deal with both grey level and binary image processing. These techniques are being usefully applied to a variety of application studies in the fields of medical prescreening, process control, and satellite data analysis. The great processing speed advantage available with this type of machine should bring many more such applications within reach of automatic analysis.

6 Acknowledgments

The authors wish to thank the following for their assistance in the preparation of this paper: Miss U. Campbell, Miss A. Harris and Miss D. Hughes.

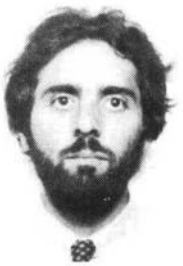
7 References

1 DUFF, M.J.B., WATSON, D.M., FOUNTAIN, T.J., and SHAW, G.K.: 'A cellular logic array for image processing', *Pattern Recognition*, 1973, **5**, pp. 229–247

2 JAMES, M.: 'A note on binomial filters'. Image Processing Group Newsletter 6, University College, London, 1978

3 ARCELLI, C., CORDELLA, L., LEVIALDI, S.: 'Parallel thinning of binary pictures', *Electron. Lett.* 1975, **11**, pp. 148–149

4 GOETCHERIAN, V.: 'From binary to grey tone image processing using fuzzy logic concepts', *Pattern Recognition*, 1980, **12**, pp. 7–15



V. Goetcherian was born in Nicosia, Cyprus, in 1954. He received his B.Sc. (Eng.) in electronic engineering from Queen Mary College, London. He was awarded his Ph.D. in image processing from the University of London after three years in the Image Processing Group at University College London. He is at present working for Logica.