DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

DAI Working Paper No. 105
Date: January 1982

Title: Parallel Image Analysis Technology

Author: Robert B. Fisher

Abstract:

This paper presents a survey of parallel processing techniques as applied to computer vision and image processing problems. Conventional processing methods are simply overwhelmed by the quantity of data associated with images. One approach to solving this problem is to process large amounts of the data in parallel. This paper summarizes work on optical, specialized parallel hardware, and specialized parallel software methods, as applied to computer vision. Also included is a discussion of the basis of the problem, the rationale for using these approaches to its solution, and an example of a hypothetical system designed utilizing some of the techniques examined.

Acknowledgements:

The author was appreciatively supported during the course of this work by a postgraduate studentship from the University Of Edinburgh. Special thanks also go to A. P. Ambler, R. Beattie, J. Hallam and J. Howe for their efforts, discussions and advice on both the content and exposition of this paper.

Table Of Contents

1. Introduction

This paper considers a major problem facing computer based vision systems, that of processing the immense quantities of data, and examines the use of parallel processing technology as a possible solution.

The term image is used here to refer to television type images, although the discussion also applies to most other image types. These include radar, synthetic aperture radar, holographic, satellite scanner (xray, infrared, ...) and sonar images. Only single image (frame) analysis is considered, although the problem is even more severe in multiple image analysis systems (motion, stereo, change detection, ...). The terms "image processing", "image analysis" and "vision" are used interchangeably, and generally mean any process acting upon an image or the results of the previous processing of an image. Hence, both raw image enhancements such as smoothing and high level operations such as object matching are considered.

The main question that one might ask is: "What is there about vision that requires (or stimulates research in) parallel processing?", to which we answer: "Most images contain too much information to be quickly processed on a standard computer.". For example, in section 2, we examine a vision application which requires an estimated $3*10^8$ operations per image. Given that standard computers run at about 10^6 operations per second, 300 seconds of cpu time are needed to process each image. Alternatively, we could use about 12000 computers to process the data at video rates. Obviously, therein lies a problem. It is largely practicality which motivates our interest in parallelism. Realtime industrial applications require reasonable processing times (under 30 seconds per image). Feedback to a robot manipulator might require times of the order of 0.1 to 1 second per image. However, even offline processing needs additional processing capabilities. In the next section, we calculate that the present ERTS satellite program requires about 250 fulltime computers to process the data collected each day. Obviously, modern computer techniques are not capable of meeting these requirements economically.

Several approaches have been used to help solve this problem. They include simplifying the problem (e.g., use of binary images), faster general purpose processors (bit-slice microprocessor technology), and faster specialized processors (covered in sections 4B and 4C). But, as we shall see in the next section, we need 3-4 orders of magnitude improvement in processing rates to meet current needs. As a result, among conventional methods, only the specialized processor offers much promise.

The application of parallel processing to vision is not new. Though the detailed workings of the human vision system are not understood, it is generally accepted that most low level processing is done in parallel. In particular, neurophysiological research has discovered two major instances of parallelism, one in the retina and one in the cortex.

The retina is composed of about 10^8 receptors, which, through the various neural interconnections, combine to form about 10^6 center-surround receptive fields (Lin72). (Though Marr (Mar74) suggests that the center-surround behavior is an artifact and the real purpose is to compute the retinex function. However, this must also be computed in parallel.) The outputs of the ganglia which calculate the fields pass directly to the brain.

The cortex has a structure which consists of about 10^6 columns of cells, each organized to correspond to specific orientations of objects (lines, edges, bars) in the visual field (Hub63). Each column consists of more than 10^3 cells of various types (simple, complex, hypercomplex). The output of cells is dependent upon inputs from a number of the receptive fields.

Based upon these values (which are themselves based upon values from the cited articles), we can estimate the retina as performing roughly 10^6 operations simultaneously, and the cortex as performing roughly 10^9 operations simultaneously. Even though biological information processing is considerably slower than electronic (10^3 "operations" per second (neural firings, Lin72, page 56) versus 10^6), parallelism still seems to provide an overwhelming improvement in processing resources.

Moreover, the nature of the problem itself is suggestive of parallelism. Obviously, the two dimensional nature of images allows spatial parallelism. That is, if we structure an image as a collection of pixels, we can process all pixels in the image independently and simultaneously. We also can achieve parallelism in time and functionality (though these categories have some overlap). Parallelism in time refers to the fact that processes may have distinct stages and systems may be built to process the stages in parallel. Functional parallelism arises when the complete image analysis task has several independent subtasks.

The existence of such models (spatial, temporal, functional) of parallelism immediately suggests techniques matched to the models. In fact, the parallel vision techniques discussed in the body of this paper all rely upon one or more of these models.

The body of this paper consists of summaries of work in optical, computer hardware and computer software techniques as applied to computer vision. The optical methods generally use lens and filter systems on the actual image prior to input into a sensor, and thus rely upon the two dimensional structure of an image. The hardware techniques discussed orient about the three models of parallelism discussed above. (We will only cover digital and CCD methods.) The software techniques discussed are algorithms suited for parallel execution. (They, of course, only achieve the potential improvements when executed in some parallel manner.) In discussing these topics, actual algorithms and implementations will be considered, as well as the general principles involved.

It is recognized that both the hardware and software depend upon the same principles (such as spatial parallelism). However, it was decided to organize the paper so as to focus on the technologies, rather than the principles.

The paper also contains a brief analysis of the computational requirements of various vision processes, and the potential benefits of each technique discussed. Lastly, it concludes with two hypothetical system designs which use parallel technology.

It should be obvious that I believe that parallel processing is a necessary part of practical vision systems. However, as no one yet knows what computations a vision system should do, many of the techniques and algorithms discussed here will never have general use. The intention of the paper is to present a description of the techniques being studied and give estimates of their potential speed improvements.

2. What is the problem? Too much data!

In this section, we examine the data overload problem in greater detail. First, there is a discussion of typical real data sources. Then, we look at what computation is required for a particular example. Lastly, we compare our analysis with other results.

First, we look at some typical data sources:

Landsat/ERTS (Lan79,Bra78)

These satellites are designed for terrestrial resource analysis and land survey. They transmit multispectral data describing the regions of the earth over which they pass. An image consists of a window of about 10^7 pixels. Each pixel consists of 24 bits of data, whose contents are 4 bands of spectral data (6 bits each). By 1978, there were 3 satellites in operation. In total, all satellites transmit about 200 pictures a day. (Though, if in full time operation, they could send about 8500).

Reddy and Hon (Red79) estimate typical image processing will require on the order of 10^3 - 10^4 operations per pixel. Here, they presumably consider mainly the computational costs of more intelligent analyses, such as image segmentation or change detection (Pri77). Because the types of processing used in typical statistical survey applications are simpler, we could use the smaller estimate, but, as the data has to be processed for each of 4 spectral bands, the higher estimate will be used.

Based on these figures, we can estimate the number of computers needed in full time operation to process the data:

```
    200 pictures/day
  * 10^7 pixels/picture
  * 10^4 operations/pixel
  / 8.5 * 10^4 seconds/day
  / 10^6 operations/computer-second (typical processing rate)
  ------------------------------------------------------------
  = 250 computers (approx)
```

Note that this figure is for 200 pictures a day. In 1981, a new series of satellites is expected to be launched, covering 7 bands of 8 bit data with roughly 7 times greater detail. This alone should present significantly greater data processing requirements. Though I have no figures for the many other types of satellites doing visual surveillance (military or otherwise), enough data will be taken each day to keep thousands of computers in full time operation.

Optical Space Surveillance

The general idea of optical space surveillance is to do telescopic scanning of the near earth region, for the purpose of object location, tracking and identification. Besides the obvious military applications, there is interest in locating the general debris in orbit. (I have heard estimates of 1000 uncataloged objects: rocks, inoperative satellites, rocket casings, etc.) In this case, the low level data processing is likely to be simpler than the ERTS processing. None the less, there are estimates that, by 1990, this application will require an additional factor of 10 in data processing capabilities over current applications such as the ERTS image processing (Nud80).

TV image analysis

In a more typical situation, we use a standard television camera as the data source. These cameras produce picture sizes in the range of 64^2 to 1024^2 pixels, with standard television about 600^2 (Ter55). Pictures are produced at intervals of 1 to 150 milliseconds, with standard television about 33 milliseconds (40 milliseconds in the UK). As we are likely to be doing more sophisticated analyses, the estimate of 10^4 operations per pixel will be used. So, our calculations for a standard television picture are:

```
   3.6 * 10^5 pixels/picture
* 10^4 operations/pixel
/ 10^6 operations/computer-second
-----------------------------------
= 3.6 * 10^3 computer-seconds/picture
```

If we want to process pictures at video rates, then we obviously have a problem, as we have to multiply this figure by 30 pictures/second. Even if we are willing to wait until the processing of a single image is complete, this estimate is excessive for industrial applications (1 hour per picture).

In any case, it is obvious that a lot of computer power is needed. We now examine a simple image processing task to see where the computation is used. In the following example (see example 1), it is assumed that the algorithms are sufficient to perform the desired task.

The steps and estimated complexity of a hypothetical solution to this example problem are given table 1. We see that the total estimated computation is on the order of 3*10^8 operations, which is roughly equivalent to 300 seconds of computation.

Nudd (Nud80) provides a diagram illustrating data rate and processing requirements of typical vision systems. It is copied below, in figure 1, with the addition of the far right column which contains corresponding values from our example computation. Our numbers in the figure are multiplied by another factor of 40 to account for a 40 pictures/second data rate (to make our figures comparable to those of Nudd, which were given for video rate analysis). The letters in parenthesis in table 1 show which operations correspond to which numbers in the figure.

Comparing the two rightmost columns, note that I give greater processing requirements at the higher levels of analysis, but that the general trends hold. However, the lowest level estimates roughly agree, and it is

these figures which dominate the computation. Of course, one would expect that the processing requirements at the higher levels would increase in relation to the sophistication of the image analysis task.

The higher levels are also the areas which will provide the greatest problem in the future. This is largely due to lack of significantly parallel models (i.e., more than a factor of 10), as well as a general lack of knowledge about the needed processing. Fortunately, at present, the greatest processing requirements are at the lowest levels, which have the greatest parallelism potentials.

This completes our examination of the image processing computational problem. The remainder of the paper concentrates upon the contributions that parallel processing technology make to the solution of this problem.

The vision task is to perform an inspection of a manufactured object, for the purpose of detecting the presence or absence of a subcomponent. Greyscale analysis will be used, because the subcomponent will lie in the interior of the object. The basic approach will be to do an edge and line analysis of the image and then match the detected lines against an image model.

Example 1 - An Image Processing Task

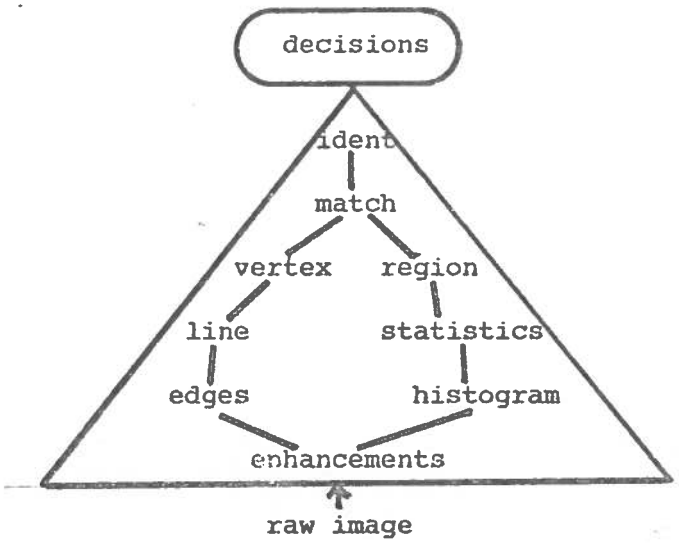
The parameters of the computation and hypothetical values are:

n - image linear dimension (512 pixels)
m - convolution window size (3 pixels)
k - number of real lines in picture (50)
l - length of real lines in picture (1000 pixels)

The three fields refer to potential steps in the process, an estimate of the complexity of the steps, and an estimate of the number of computer operations needed to execute the steps. The letters in parenthesis (a,b,c,d) are for identification with processing stages shown in figure 1.

Step	Complexity	Operations
preprocessing convolutions	10 * m^2 * n^2	2 * 10^7 (a)
edge mask operations	100 * m^2 * n^2	2 * 10^8 (a)
threshold operation	10 * n^2	3 * 10^6 (b)
edge thinning	100 * n^2	3 * 10^7 (b)
edge tracking	500 * l	5 * 10^5 (b)
line finding	20 * l * log2(l)	2 * 10^5 (c)
line parameter extraction	20 * l	2 * 10^4 (c)
line joining/vertex location	100 * k^2	2 * 10^5 (c)
model matching	100 * k^2	2 * 10^5 (d)

Table 1 - Analysis Of Sample Image Processing Task Of Example 1



	Data Rate (samp/sec)	Throughput (inst/sec)	Example Throughput (inst/sec)
ident	10^2	10^3	10^7 (d)
match	10^3	10^4	10^7 (c)
vertex region	10^4	10^6	10^9 (b)
enhancements	10^7	10^9	10^10 (a)

Figure 1 - Nudd's Analysis Of Image Processing Requirements (Nud80)

The rightmost column was added by the author. It refers to the estimated instruction throughput for the example analyzed in table 1. The letters in parenthesis relate stages in this figure with steps in the analysis.

3. Optical Processes

While optical methods were the earliest parallel image processing methods, it is probable that they were not generally thought of as such. Parallelism associates conceptually with a set of structured independent entities, whereas optics tends to consider images as wholes. None the less, if we accept that an optical process is performed all points of an image simultaneously, then we see that both views hold.

Principle

Consider the image as a parallel optical wavefront and then use standard optical methods to process all portions of the wavefront simultaneously.

Optics, of course, deals with light and the images we currently consider originate as such. They may be natural, such as from the normal world, or artificial, such as from photographs, slide projections, television or computer monitors.

The optical methods we consider rely upon lens and filter systems used with coherent light (laser) sources. The major feature of such systems is that, with certain lens configurations, one can create an image that is the two dimensional Fourier transform of the original image. This new image allows the application of simple operations, such as masking, which have complex effects when the original image is reconstituted. Typical applications include smoothing, gradient edge masking and matched filtering (Sta75a,Lee73).

There are other optical methods not classed with the above techniques. These include various polarized light methods such as phase contrast microscopy and defocussing (Tip68). Further, the techniques considered here are not limited to just optical frequencies. Electron beams have similar properties and one application of this is mentioned below. Optical techniques can also be used for the processing of reconstructed holographic and synthetic aperture images (Koc75). The books of Soroko (Sor80), Casasent (Cas75) and Tippett et al. (Tip68) discuss the general theory and application of optical image processing techniques.

A typical setup for an optical processing system is shown in figure 2 (Hus73).

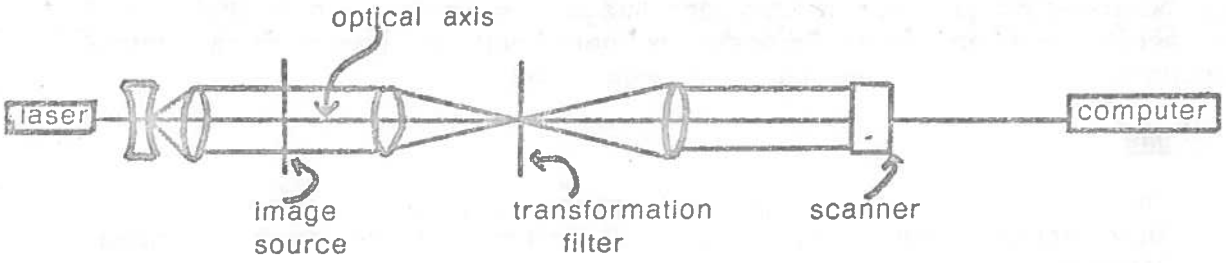


Figure 2 - Typical Optical Processing System

In this setup, the processing occurs as follows: A laser is defocussed to produce a parallel beam. This beam passes through the image source, say a photographic negative. This image is then focussed onto the transformation filter plane, where the desired filter is applied. In a reverse process, the image is reconstituted and passes as input into the scanner. The output of the scanner, the digitized image, then passes into the computer for further processing.

The major technical principle involved here is that if the input light is from a coherent source, then the focussed image, in the transformation plane, is the two dimensional Fourier transform of the image (Scr80). At this location, we can easily apply filtering operations to the image. For example, a simple high pass filter, as defined in the spatial frequency domain, is:

$$H(u,v) = \begin{cases} 0 & \text{if } u^2 + v^2 < R^2 \\ 1 & \text{otherwise} \end{cases}$$

where R is the filter parameter, and u and v are the rectangular coordinates of the filter plane. (Polar coordinates would be more useful in this case.) This filter removes all low spatial frequency components from the image. In practice, it can be implemented as a circular mask designed to block out all light inside a radius R about the optical axis.

If a photographic film is used, rather than just a mask, as the filter plane, then we can implement more general linear filters (Lee73). The film allows one to appropriately scale the intensity values at each point in the transformed image. The general form for this processing is:

Let F be the Fourier operator, and F' the inverse operator.
Let H be some filter as defined in spatial frequency domain.
Let I be the image as defined in image domain.

Then,

Output = F' (H * F(I)) (multiplication in frequency domain)

or equivalently:

Output = F'(H) ⊗ I (Convolution in image domain)

Notice that the alternative formulation above is that of a convolution. Hence, the process we perform here is the analog equivalent to the convolution masking operation when applied to digitized images (Bra65).

The types of operations that one can perform using this technology are generally limited to those linear operations based on masks and filters. Lee (Lee73) gives a good discussion, with examples, of typical operations using these components. They include:

image smoothing (low pass filtering)
image sharpening (high pass filtering)
gradient edge finding
general image arithmetic
texture (variance) analysis
image restoration
matched filtering (spatial autocorrelation)

The last item on this list, matched filtering, deserves some special mention. Suppose we wish to search images for instances of a reasonably simple object, such as machined part against a conveyor belt. We could use a matched filter that is the Fourier transform of the image of that object. Such a system will enhance image points corresponding to the desired object in the image, and hence direct the application of simpler processing. Examples of the use of this technique are corner detection, fingerprint matching, and alphabetic letter detection (Lee73, Sor80). Unfortunately, the method has problems which include:

1. All image points are affected according to the degree of match between local spatial frequencies and those of the desired object. As a result, the technique only gives indications of regions whose frequency characteristics are similar to the desired object. This is considerably different from actually locating instances of the object.

2. The technique's effectiveness varies in relation to the simplicity of the spatial frequency of the object. That is, the simpler the object, the more pronounced the results of this process (Lee73). Correspondingly, the more complex the desired object, the less obvious is its

detection among other partial matches.

3. The technique requires one filter for each type of object desired. Hence, when searching for several objects, new filters would have to be mechanically switched into the focal plane. This switching can severely degrade performance, as it introduces a new serial element into the processing.

4. The filters are not rotationally invariant. This means that a separate filter would be needed for (roughly) each potential orientation of the desired structure.

The obvious benefits of the optical technique are that with such a system one can instantaneously perform a set of operations over a whole image. There are also many problems associated with the technique. The major ones are physical, and these are discussed below. Another problem lies in the nature of the processing, which is largely content independent (except for the marginally better matched filter technique). This is because the techniques depend mainly upon the physical structure of the image, than upon the content of the image. Hence, it is hard to take advantage of what might be known about the objects contained in the image (which is part of what allows human beings to so effectively process images). In the case of optical processes, we are limited to general image properties, such as what an edge might look like (as compared to what a change in intensity might correspond to). The matched filter technique is a bit more sophisticated in these respects, but its performance is not as high as desired.

The other problems relate to the physical properties of the system. The first item to note is that the technique depends upon a coherent light source. This leads to the requirement that the image must somehow be introduced by filtering a laser beam, as seen in figure 2. This limits us, at present, to modifying the intensity of the beam via the use of masks or photographic images, with the attendant mechanical problems associated with inserting the images into the path. This also degrades the speed of the technique. A liquid crystal device (LCD) has been invented (Jac73) which allows an external image to dynamically create a mask. The image to be processed enters the system as a incoherent (natural) beam and impinges upon the device. The action of the device is such that it creates a mask corresponding to the regions of high intensity in the image. The coherent beam then shines through this mask, to create a binary image. A similar technique is used where the image (already digitized) is written onto an electronic grid, which then filters an electron beam (Cas75). The resulting beam is treated in a manner analogous to the optical processes discussed, and then redigitized at the end. This technique allows the use of greyscale images. The techniques require on the order of 0.1 and 0.03 seconds respectively per image to form the masks.

Another limitation is that the coherent beam technique can only process binary or greyscale images but not multispectral (ie, color) images.

Lastly, there is the problem of the filter design and implementation. Though the physical and mathematical principles involved are relatively simple (convolutions and masks), the selection of actual filter types and how to generate them appears to be more of an art than a science. Lee (Lee73) gives a discussion of some techniques used. Further, requirements for the dynamic selection of masks leads to a similar problem to the image introduction problem discussed above.

This completes our discussion of optical techniques. In the next section, we discuss how some interesting variations in computer architecture can be used for computer visión.

4. Computer Based Hardware

In section 1, three modes of parallelism were introduced: spatial, temporal and functional parallelism. This section considers innovative hardware structures which capitalize upon each of these three modes. In subsection 4A, we look at array structured computer systems, whose processor organization corresponds to the spatial organization of the image. Section 4B examines what are generally known as pipeline systems, in which the parallelism lies between the different stages of the image processing. Lastly, section 4C considers systems based upon functional parallelism, whereby independent subprocesses can be executed in parallel. K. S. Fu (Fu78a) also overviews many of the systems reviewed here.

A point which applies to all of the techniques is that parallelism also needs to be considered for the storage of the image data. (Otherwise, the parallelism of processing is defeated by the queueing for access to the memory.) VanVoorhis (Vor78) discusses the design of practical memory structures for parallel processing.

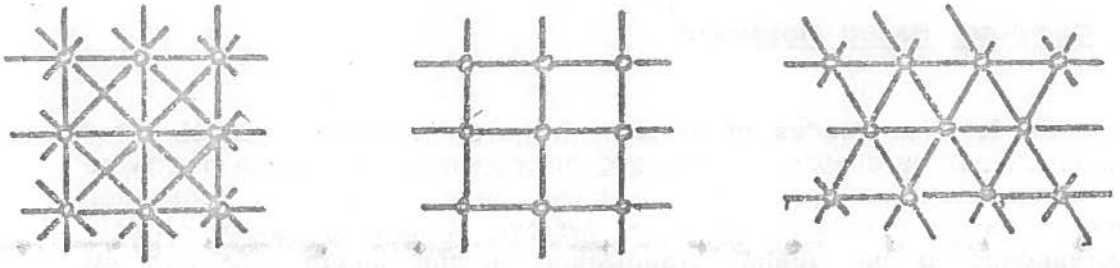
4A. Array Structured Computers

One of the most noticeable features of most image representations is the regular two dimensional structure. In most instances, this structure is a rectangular array (figure 3a), although M. Golay and others (Gol69, Pre71) have made arguments for hexagonally structured arrays (figure 3c). The use of square arrays predominates, perhaps due to the use of raster scans in television camera output.

Principle

> Use system architectures whose structure corresponds to the structural parallelism inherent in digitized images representations.

The regular parallel structure of the image arrays is highly suggestive, especially when considering the image processing algorithms which are based upon the structure. One example algorithm is a noise removal smoothing process, which, at each pixel in the image, sets the value of the pixel to the median value of intensities from the neighborhood about the pixel.



(a) 8 connected square array

(b) 4 connected square array

(c) hexagonal array

Figure 3 - Image Array Structures

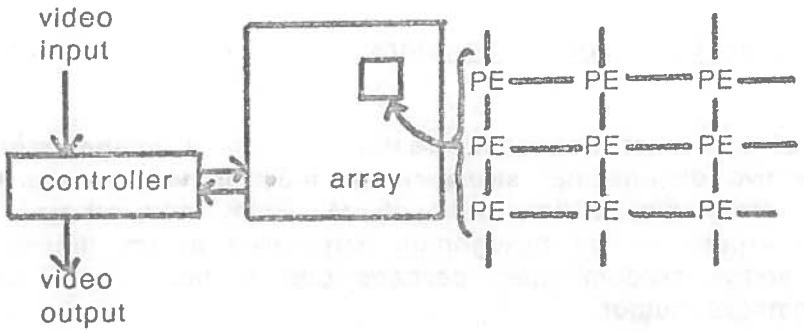


Figure 4 - A Typical Array Structured System

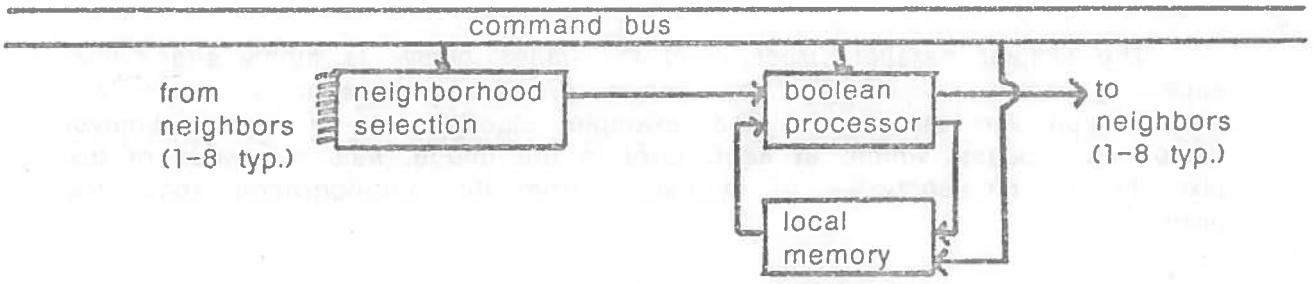


Figure 5 - A Typical Processing Element

It would be ideal if we could execute this algorithm at all of the pixels simultaneously. This requires some sort of processor for each pixel, which is the basis for the design of the large array structured processor systems. A typical structure is shown in figure 4, in which the computer system consists of a network of simpler processors connected in a two dimensional array. Each of the separate processing elements (PE) is an independent microcomputer, and is connected directly to its eight neighbors. Each of the PEs is used to store a pixel from the image, with neighboring pixels stored in neighboring PEs. (Large images can be decomposed into separate subimages which are then stored in parallel in the PE array.) In addition to the array, there is a standard computer, which acts as a controller for the array. The controller, at appropriate times, decides upon instructions to be executed by the array. It then sends out the instructions on a common command bus. The controller processor is also used to do the more routine computations, such as index calculation and program control.

Each of the processing elements is typically a very simple microcomputer. A model based upon the CLIP-4 processing element (Duf78) is shown in figure 5. It consists of three major components: a boolean processor, a local memory and a neighborhood selection unit.

The local memory contains data values for the pixel(s) associated with the processor. They may include raw image intensity values (binary or greyscale), results of processing or alternative images.

The boolean processor executes one of the sixteen binary boolean functions upon two single bit data elements, which may have come from either the neighborhood or the local memory. All PEs in the array execute the same function during each processing cycle.

The neighborhood selection function allows the PE to include the values of selected neighbors as input into its calculations. The selection of which neighbors to access is by program control, and the data from those neighbors is typically 'or'ed together when more than one neighbor is selected.

Notice that the boolean processor has two outputs, one to its local memory and the other to its neighbors. This feature can be used in dramatic ways, especially if the two outputs are calculated by different boolean functions. If the output propagated to the neighbors is also a function of the input from the neighbors, then computations may propagate across the whole array in a recursive fashion. (One use of this feature would be for region labeling, in which the label of a region is propagated into all appropriately connected neighbors of previously labelled pixels.) Of course, this complicates the architecture, which may now have to wait an indeterminate amount of time for a single instruction to complete. There is also the problem of non-convergent computations (Bla81).

Reeves (Ree80a) discusses a systematic approach to the design of such arrays and their instruction sets, along with an analysis of the 3 classes of instructions typically executed: boolean, near neighbor and recursive near neighbor.

One problem with this architecture is that a PE does only single bit boolean functions hence is best suited for binary image calculations. Fortunately, greyscale images can be stored in the local memory and the

desired functions executed one bit at a time. The programming of such greyscale functions can be tedious, but in practice need only be done once, with the resulting code reusable in subroutine form. Of course, it is the general program capabilities of the control processor which allows this convenience.

A second problem with such arrays is intrinsic, in that the computations of each PE are based upon local information, whereas some processes need global information (such as threshold selection). Reeves (Ree80b) makes some suggestions for simple architectural enhancements to help alleviate this problem.

A practical problem is that it is not yet economically feasible to dedicate one PE to each pixel, as typical images consist of 256^2 to 1024^2 pixels. (Though VLSI may change this soon.) At present, the largest array has only 128^2 PEs (see table 2). Hence, it is necessary either to process only a subset of the original image, to process the whole image in stages, or to store multiple pixels at each PE. For the 1024^2 image, this can be done by subdividing the image into 64 windows of 128^2 pixels each. Then each PE would have stored with it the data from the corresponding pixels in each of the 64 windows. This complicates the programming, as boundary effects now need be carefully considered (which may suggest alternate organizations).

A feature of such arrays is that it may be possible to select, under program control, either 4 neighbor rectangular, 8 neighbor rectangular or 6 neighbor hexagonal connectivity, as illustrated in figure 3. Further, the connectivities of the edge PEs may be selectable, as in the MPP (see below), which allows open, cylindrical, spiral (all PEs in 1 chain) or circular (all PEs in a loop) arrangements.

Algorithms suitable for implementation on these machines are discussed in section 5A. The remainder of this section comments upon the actual array structured machines of the type described above, and a few others of somewhat similar character.

In table 2, we summarize the major features of three array machines: the ICL Distributed Array Processor (DAP) (Mar80), the University College of London Cellular Logic Image Processor (CLIP) (Duf78), and the Goodyear Aerospace (in conjunction with NASA Goddard Space Flight Center) Massively Parallel Processor (MPP) (Bat80). Among the CLIP machines, only the fourth in this series of research machines, CLIP-4, is included. Duff (Duf77, Duf73) discusses previous machines and reference (ano81) briefly mentions possible future machines (CLIP-5: improved performance, CLIP-6: more powerful PE).

machine	date built	array size	local memory (bits)	instruction time (usec)	connectivity
DAP(pilot)	1979	32*32	1024	.2	4
CLIP-4	1978	96*96	32	10	4,6,8
MPP	1982(est)	128*128	1024	.1(est)	4

Table 2 - Array Structured Computers

Previously, we discussed storing multiple windows of an image in a single PE. In the MPP, then, the 1024 bits of local memory would allow 16 bits per pixel when processing a 1024^2 picture. This is a reasonable number for many image processing functions. The CLIP-4 machine is more restricted in that it has only 32 bits of local memory.

The primary advantage of this architecture lies in the obvious degree of parallelism. The CLIP-4 machine has about 10,000 processors, hence can provide an extraordinary degree of improvement in suitable applications. Cordella et al. (Cor78) estimate the improvement factor (the ratio of machine cycles) gained by implementing various algorithms on such an array. The range of ratios for typical operations (for a 1024^2 array) is:

operation	ratio
thresholding	10^2
perimeter counting	10^3
smoothing	10^5
contour extraction	10^6
thinning	10^7

Even if we consider the slower cycle time of the CLIP-4 machine, about a factor of 10 over standard processors, the degree of improvement can be substantial. However, not all algorithms are well suited for this architecture (such as FFT algorithms). Further, these ratios are only for the actual algorithms, and do not include the loading of the image into the array (video rate) and the unloading of results.

Other similar machines (Lov80)

One machine which is conceptually similar to the above is the Goodyear STARAN machine (Pot78). This machine was designed about the

concept of a collection of associative processors, which execute the same program synchronously, but only in the processors whose memory contains specific patterns. A full machine consists of up to 32 processor banks, each of which contains 256 processors. Each processor has 9216 bits of local memory. A 512*512 image with 8 bit pixels can be processed in one processor bank.

The advantages of this machine come from the fact that all 256 processors in a bank can execute simultaneously, as in the case of the two dimensional arrays discussed above. However, as the structure of the arrays is linear, they require cleverer image storage and processing algorithms. The processors, on the other hand, are more like normal processors (ie, not simply 2 input 1-bit boolean units), hence can give improved performance. Rohrbacher (Roh77) discusses the application of this machine to several image processing tasks. These include convolutions (3*3 over 512*512 in 0.6 second), image warping (3-7 seconds) and 2 dimensional FFTs (2-9 seconds). These figures show an improvement of about 100 over conventional machines. Moreover, it is more difficult to program a FFT algorithm on one of the array systems discussed previously, due to the local connectivity of the PEs. (The STARAN FFT application required the addition of a FLIP permutation network, which allowed each cell to access any other's memory, in an orderly manner). Siegel (Sie81b) gives a description of a 2D FFT algorithm which gives an O(n) improvement over a linear machine, when implemented on a n*n array.

The Associative Linear Array Processor (ALAP) (Fin77) is an associative processor similar to the STARAN machine.

Another class of machine includes the ILLIAC-IV (Bar68) and an unnamed Japanese research machine (Mat79). These machines consist of arrays of processors organized into a square of size 8*8 (or 64 linear elements) and 4*4 respectively. All machines execute the same instructions synchronously, in a manner similar to previously discussed machines. However, in this case, the PEs themselves are considerably more powerful, and execute significant arithmetic operations. Again, the architectures are most useful for local computations.

The Purdue Multi-Mode Multi-Microprocessor (PM^4) (Bri79) system is a bit more exotic in that it is designed to execute either the same instruction over the whole array or different instructions. It can be dynamically configured into organizations with groups of processors executing independent processes and accessing different blocks of memory. The number of processors is of the order of 256. The PASM system (Sie81a) has similar features.

Also in this category comes the Pepe machine (Parallel Element Processing Ensemble) (Vic78). This machine consists of a CDC 7600, 3 control units and up to 288 parallel processing units, each composed of 3 processors (data correlation, arithmetic and associative output). A 36 unit version was delivered in 1975. While designed mainly for ballistic missile defence systems, applications to image processing are also considered.

The last architecture considered is more speculative. The general idea is to include some processing elements directly in the retina of a CCD camera (Nud80), in a manner analogous to the human retina. Thus, it would be conceivable to have a camera whose output was Walsh transform detected edges, rather than image intensities. The nature of this processing would be similar to that of the arrays considered above, but the

algorithm would be permanently selected. No working examples of such retinas have been reported.

4B. Pipeline Configurations

Considering the example of subcomponent detection analyzed in section 2 (example 1), we notice that the processing of the image occurs in distinct stages. In this case, there are nine stages, from preprocessing to model matching. Each of these stages is virtually independent of the others, in that the execution of each is dependent only upon its input data, which arrives from the previous stage. If one could configure a processing system which allowed each of the stages to execute in parallel, then we could again expect speed improvements. The factor of improvement would be limited to the degree of parallelism achieved, which, with pipeline systems, is likely to be only of the order of 10.

Principle

> Use processing system architectures whose structures correspond to the sequential structures inherent in the processing.

In discussing this topic, we refer to figure 6, which is a hypothetical pipeline system performing the component inspection task analyzed in section 2. The system consists of five processors, the first four of which do the first four stages of the processing analyzed in table 1. The fifth stage does the remaining steps (5-9). Each stage executes its process in parallel with the other stages.

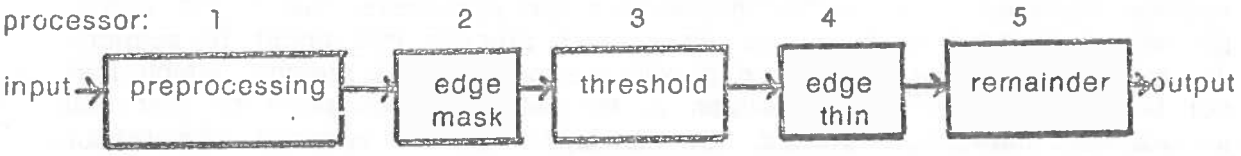


Figure 6 - A Pipeline Structure

Such a structure has been given the name "pipeline". This name refers to the fact that data flows between the processes, rather than the program control changing to execute new processes on the same data. Cooper (Coo77) discusses the construction and analysis of pipeline structured systems. Patel (Pat80) criticizes Cooper's distributed pipeline approach and suggests a breadth parallel structure rather than sequential structure. Unfortunately, this suggestion would require each processor to be capable of the full processing task, unlike Cooper's pipelines. The pipeline LSI chips mentioned below are only suitable for high performance of particular simple processes, and hence are not suitable for Patel's suggestion.

There are two major variations of this structure, relating to the nature of the processing and the timing of the data transfer between stages. In one variation, full images are processed at each stage, in parallel, and the results are transferred to the next processor. In the other, the image is processed as pixels arrive and which are then sent immediately (or with a slight delay) to the next processor.

Both approaches have advantages and disadvantages. The block transfer method allows more complex processing, in that it has the whole image available, but requires substantial memory for image storage, and lengthy image transfer times. The transfer can be overlapped with

processing at the expense of additional storage for i/o buffering. The continuous flow method requires considerably less image storage and can overlap data transfer with processing. The reduced data storage limits the algorithms implementable to those which use only data which has arrived relatively recently. In most configurations, this means just the most recent rows of pixels, hence the algorithms must be just local neighborhood functions. (However, the windows may still be reasonably large, as in the 26*26 case mentioned below). Another requirement is that the time to process one data element must be, on the average, less than the time until the arrival of the next data element.

The pipeline shown in figure 6 is strictly linear, although there is no requirement for this. For example, the output of the preprocessing stage could also have been used as input into a region based analysis pipeline.

One general problem with pipelines is that, in order for the full benefits of the parallel structure to be obtained, each of the stages in the pipeline must be 100% utilized. Unfortunately, most processing stages are not equally complex, as in the case of our example. Referring to table 1, we see that we need approximately 10^7, 10^8, 10^6, 10^7 and 10^6 operations at each of the 5 stages. So, if all processors were equal, the average utilization of the five processors would be: 10%, 100%, 1%, 10% and 1%, which is clearly a less than ideal situation.

One solution to this problem is to make the processors which do the most computations be the fastest (trading cost or complexity for speed). This might require customized processors, such as those discussed below. Another approach is to further decompose the processes, but in the example we would need to divide the edge mask process into about 10 sequential substeps in order to balance it with the next largest process, which may not be possible. A third approach is to make all processors so fast that all are less than 100% utilized. (Or conversely, to use reduced data rates.) That is, the processors will be able to process the incoming data even at the maximum data rate. This approach is also used in the devices discussed below. A fourth approach is to place multiple instances of the slow stages in parallel and to use multiple buffers. An example of this structure is shown in figure 16.

Even if we were to balance the rates of all processors, then this would still allow only a factor of 5 in parallelism. (The unbalanced pipeline achieves 1.2). In consequence, we argue that the parallelism benefits here are not significant.

The remainder of this section discusses hardware structures which were designed or built in accordance to principles such as these. A common feature is that all rely upon the approach of processing the data on a pixel by pixel basis.

Nudd (Nud80) reports upon a series of experiments using CCD technology to produce single chip processors capable of doing image processing functions at video data rates. The advantage is that this is currently the maximum data rate from standard television cameras (approximately $7*10^6$ bits/second), hence, this image processing can be done in real time. Thus, using a series of such chips, one could easily and cost effectively do a majority of the low level image processing. In the pipeline of figure 6, the first four stages do 99% of the computation. Each of the four processes are suitable for single chip implementations.

Table 3 summarizes the performance of the devices discussed by Nudd (Nud80). Of note is the programmable 5*5 convolution window chip. Because of its size and general programmability, it would be suitable for a variety of filtering operations, including smoothing and edge enhancement.

Device	Data Rate	Functions
1	5 khz	edges, hp filter, Laplace
2	2 Mhz	Sobel, means, unsharp mask, adaptive stretch
3	7 Mhz	Laplace, 5*5 programmable convolution, median filter, bipolar convolution

Table 3 - Pipeline Chip Prototypes
(Data from Nud80)

Of special note is a 26*26 bipolar convolution window which executes at video rates. The function of this processor is to calculate a circularly symmetric weighting function, which is much like a local gradient function. In some respects, this chip mimics the center-surround receptive field calculation of the retinal ganglia (Lin72). The processor was designed to provide the input for a "primal sketch" processing system based upon the work of D. Marr at MIT (Mar77).

It should be noted that the video rate processing components give a remarkable amount of performance improvement - perhaps a factor of 1000. This performance is largely due to the specialized nature of the processors. The parallelism itself might only provide another factor of 10, presuming there were 10 functional units in the pipeline.

Brabston et al. (Bra78) discuss some of the design issues of a pipeline system for processing Landsat data. Stages include geometric warping, radiometric correction, image enhancements and format conversion.

Roesser (Roe78) discusses the design of a speculative two dimensional pipeline system, in which the results of the computation flow across a square processor array in both horizontal and vertical directions. The image points enter the array at the edges. The type of processing suggested for this architecture is based upon a "two dimensional generalization

of the common state-space model for linear time-invariant discrete dynamical systems" as applied to images (space-invariant). Kung (Kun80) reports on research on the systolic array concept, whereby data flowing through a pipeline of simple synchronous processing units is eventually transformed. Reverse flows of intermediate results is utilized. This type of processing is performed at the lowest levels of the image.

The GOP system (Gra81) uses four parallel pipelines to give fast programmable convolutions. The DIP-1 system (Ger81) allows flexibly programmed pipelines, reconfiguring a small set of simple processing units under micro-program control.

This completes our analysis of pipeline structured systems. In the next section, we look at structures in which the parallelism is due to the ability to organize the subtasks of the image processing task into parallel independent processes.

4C. Functional Parallelism

Hanson & Riseman (Han78) give a description of image segmentation based upon a combination of edge and region analysis. The initial data for both of these processes is the same (the image), but is processed in separate computations, with the results eventually merged. Ideally, as the actual processes are independent, they could be executed on separate processors.

At a lower level, a number of individual image processing operations are identifiable as being common to a variety of image processing tasks. These include convolutions, local window logical operations, and histogram calculations. This has prompted researchers to develop special purpose function units capable of executing these functions at high speed. The units would operate as auxiliary processors attached to a main computer. They can then be used to do the desired special functions in parallel with each other and the main computer. This leads to:

Principle

> If the problem structure has independent subtasks, and they are suitably definable as distinct functions, then configure the hardware to execute these subtasks in parallel.

The two systems discussed in this section consist of a main processor and one or more attached function units. From the descriptions of the systems, it appears that the units can execute in parallel with each other and the host, but this capability is not generally used. In these cases, the performance of the special purpose unit is of much greater significance than the minor amount of parallelism they provide, which is comparable with the pipeline systems discussed previously. As a result, they are only described briefly.

The general system structure is:

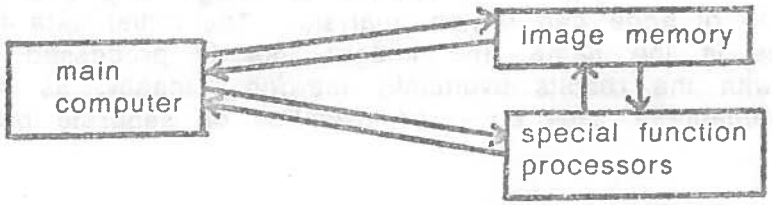


Figure 7 - Functionally Parallel System Structure
(Each unit executes in parallel with others)

TOSPICS

The Toshiba Pattern Information Cognitive System (TOSPICS) has been described by Mori (Mor78) and Fu (Fu78a). This system was oriented towards the application of image processing to medical and remotely sensed data. The system consists of 4 frame stores, each consisting of 512 2 8 bit pixels, a local parallel pattern processor (PPP) functional unit, and a Toshiba TOSBAC-40C host computer. The PPP consists of six or seven functional subunits. The subunits and their associated functions are:

convolution - up to 8*8 programmable windows
logical operators - used for thinning, shrinking, boundary detection
coordinate transformation - as in image translations and rotations
region labeling - according to programmable connectivity modes
data conversion - 8 bit table driven mapping, for thresholding, histogram equalization, log compression
pixel operations - general microprogrammable capabilities for non-linear coordinate transforms or texture analysis, etc.
histogram generation - of image greylevels

The units are designed to give a performance improvement of roughly 100.

PPM and PICAP

The Parallel Picture Processing Machine (PPM) and PICAP systems, as described by Kruse (Kru73,Kru78) and Fu (Fu78a), were designed and built at Linkoping University, Sweden. The PICAP machine is a more recent and advanced version based upon the original PPM design. The goals of this development were to develop a system which could apply local operations to an image at high speed. The system has been applied mainly to analysis of fingerprint and malaria parasite images.

The system consists of 9 frame stores, each consisting of 64 2 pixels of 4 bits each. The system uses a standard minicomputer host and has one special functional unit.

The functional unit consists of 9 subprocessors, which can be

configured as either a 9-vector or as a 3*3 array. The 9-vector mode is useful for processing multiple images. The unit executes two classes of functions when in the 3*3 configuration. The first class is the standard 3*3 programmable convolution window process. The second is a bit unique. The programmer can define a sets of logical relations on the window. When the relation holds, then the value of the center pixel is changed. This process can be used for neighborhood counting, positional determinations, and shrinking.

The IA system (Lan81) has special units to do boolean operations over complete image planes, and a variety of mask based operations, all on a hexagonal grid.

This processor has an improvement factor of about 20.

This completes the discussion of functional parallelism, and the other hardware techniques. In the next section, we overview the general classes of algorithms whose structures are suitable for parallel implementation.

4D. Other Parallel Machines

We briefly mention here several other image analysis machines. These machines are designed with parallelism viewed at a system level rather than a task level. The FLIP system (Gem81) is a collection of 16 microprocessors, which can be configured under microprogram control to cooperate as pipelines or as cascades, all in the execution of a single function (instruction). The EMMA system (Man81) consists of groups of micro-processors (60-70) organized as families connected by several data busses. The SYMPATI system (Bas81) partitions its image memory into 16 parallel blocks and associates a processor with each block. This allows 16-parallel execution of many local-neighborhood functions.

5. Structurally Based Parallel Algorithms

In section 4, we looked at hardware structures developed to execute parallel algorithms. We now look at the algorithms themselves.

One class of algorithms are those whose parallelism is due to their computational structure. The pipeline based and functionally parallel processes used as examples in the previous section have such parallelism. They are characterized by having distinct, well defined subfunctions, which can be structurally isolated as subcomponents of the total process. This form of high level parallelism is thoroughly studied as a subject of computer science (Bri77) and is not discussed further.

Another class of parallelism is algorithmic parallelism. When examining an algorithm in detail, we find that many of the computational steps are independent of other nearby (in the algorithm) steps. Data flow machines (Wat79) have been designed to capitalize upon the potentials inherent here. This is also a subject for computer science, and is not discussed here.

The algorithms discussed in this section are those based upon the parallelism that occurs in the image data structure, that is, the regular array of image points. In section 6, we look at algorithms whose parallelism is based upon the contents of the images.

This section has 2 subsections. In the first, we look at the algorithms which can be applied over the whole image array. The second subsection considers the special class of the relaxation algorithms. (These algorithms are also applicable to the approaches of section 6, but will be mainly discussed here.)

5A. Image Array Algorithms

In section 4A, we discussed the features of parallel array machines. These were designed with the intention of executing the types of algorithms discussed in this section.

An image is generally represented as a two dimensional array with either square or hexagonal cells. Many operations which are typically done to an image, especially at the lower levels of processing, only use data concentrated at a pixel and some regularly defined neighborhood about that pixel. Because of these two features, it is both feasible and desirable to execute an operation simultaneously at all of the pixels in the image.

Principle

Apply local neighborhood processes in parallel at each suitable neighborhood in an image.

An example of an array operation is a local averaging operator, as illustrated in figure 8. The operator produces, as the output value at each pixel, the average of all pixels in a local 3*3 neighborhood (for example, about the circled pixel below). This is formed by multiplying the values of the pixel (in the neighborhood) by the values in the mask, summing the results, and dividing by an appropriate scale factor (here 9). This is an instance of a more general class of algorithms based upon the convolution process.

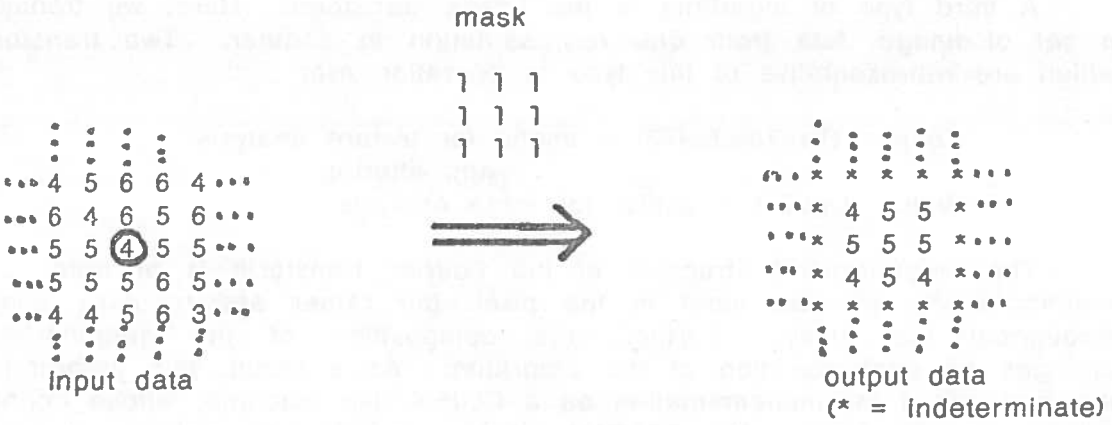


Figure 8 - A Smoothing Operator
(The operator works by averaging the points in a region according to the values in the mask. In particular, the center pixel is averaged to the value 5.)

Convolution operations consist of applying a window, such as the above, to each pixel in the image. The shape and coefficients of the window are chosen according to the task. Some uses of this algorithm are:

smoothing (Cor78) - as above
spatial frequency filtering and other mask operations - for edge, region or texture analysis
Sobel operator (Nud80) - edge detecting

If we allow the results of computation at each pixel to propagate to neighboring pixels, then we have the basis for algorithms affecting the whole array. In this instance, the termination of the algorithm may be either explicitly determined or rely upon convergence of the computations at the pixels. The propagation of results can be either omnidirectional or selective, depending upon the desired effects. Applications of this technique include:

region segmentation and labeling - region labels propagate to appropriate neighbor regions, according to connectivity constraints.
background removal (Duf76) - connectivity applied to the irrelevant portions of the image.
line completion (Wal78) - line designations propagate in the line direction, hoping to connect segments with gaps.
symmetry detection (Wal78) - markers propagate in directions conducive to symmetries, hoping to meet other such markers.
perimeter finding (Duf73) - finds the outside edge of a region

A third type of algorithm is the image transform. Here, we transform a set of image data from one representation to another. Two transforms which are representative of this type of operation are:

Fourier (Tan78a,Baj73) - useful for texture analysis and filtering
Walsh (Ogo76) - useful for edge analysis

The neighborhood structure of the Fourier transform is of note. The neighborhoods are not local to the pixel, but rather are regularly spaced throughout the array. Further, the composition of the neighborhoods changes on each iteration of the algorithm. As a result, this algorithm is not well suited for implementation on a CLIP-4 like machine, whose connectivities are just local. The STARAN machine solves this problem by use of a clever neighborhood shuffling device called the FLIP network.

Lastly, we include a collection of neighborhood operations which don't conveniently belong to the above classes. They are representative of the variety of operations possible using parallel array techniques. They include:

thresholding (Cor78) - set all image intensity values according to a local or global threshold.
edge thinning (Arc75) - reduce edge cells detected (as output of some edge operator) to thinner edges.
silhouette smoothing (Skl76) - smooth the outlines of objects.
noise cleaning (Sta75b) - apply non-linear operators to reduce the image intensity variance.
contour extraction (Cor78) - border following
skeleton finding (Duf77) - finds the skeleton describing a region
concavity detection (Skl76) - finds concavities in boundaries of silhouettes.

The majority of algorithms discussed above use a rectangular array structure for representing the image. Golay (Gol69, Pre71) has suggested algorithms based upon a hexagonal grid structure which eliminate some connectivity and distance problems.

All of these operations are very low level, in that they act directly on the image intensity array, and make little use of the content of the image. This other knowledge may be in the form of:
consistency relations that must hold between neighboring pixels, as a consequence of image structure,
higher level consistencies due to structures in the image (such as edges),
and external knowledge of potential image contents.
The next section considers a class of algorithms which address all of these points.

5B. Relaxation Techniques

In the previous section, we looked at algorithms which made little use of any relationships which held between neighboring pixels. This implies that these algorithms are not using any higher level knowledge to extract the information in the image. To begin with, there is the physical consistency of the scene, that is, surfaces are smooth and have consistent textures, lines are continuous and intensities vary gradually. Of course, such statements are only generally true, and the less homogeneous the scene is, the more variations we encounter. Further, there are scale dependencies in the notions of continuity, consistency and graduality.

We also have knowledge available to us, as humans, that allows us to interpret what we see. There are high level facts, such as the shapes of expected objects. We also know a lot about how objects relate to images. We can interpret a line as an object boundary, with object to one side and background to the other. This might suggest interpretations of shadowed or partially occluded objects. Further, we might know that the boundary must be a particular boundary, or at least one of a smaller set. In all cases, what we know in addition to the actual image is substantial and, if it were possible to apply such knowledge, would greatly enhance image analysis.

On the other hand, when we apply an operator to a raw or previously processed image, we generally use only local information from the image. This is partly due to the computational difficulties associated with the large number of pixels in an image, and partly due to our inability to design conceptually meaningful operators (which act on more than just simple data patterns). Because of their locality, these operators are sensitive to both noise and ambiguity. (Of course, some processes should only be done locally, as the contents of most locales are independent of all others.)

The class of relaxation algorithms has been designed to add some of the higher level image consistency information to the processing of locally based information. They have been used for both low level (pixel based) and higher level (object based) processing (Kit79,Hum78,Zuc78b). The general idea is that neighboring elements should have some consistency between themselves, where the type of consistency depends upon what the elements are considered to represent. If models for acceptable modes of consistency can be formulated, then the models can be used to reduce the ambiguities of interpretation (Zuc78a).

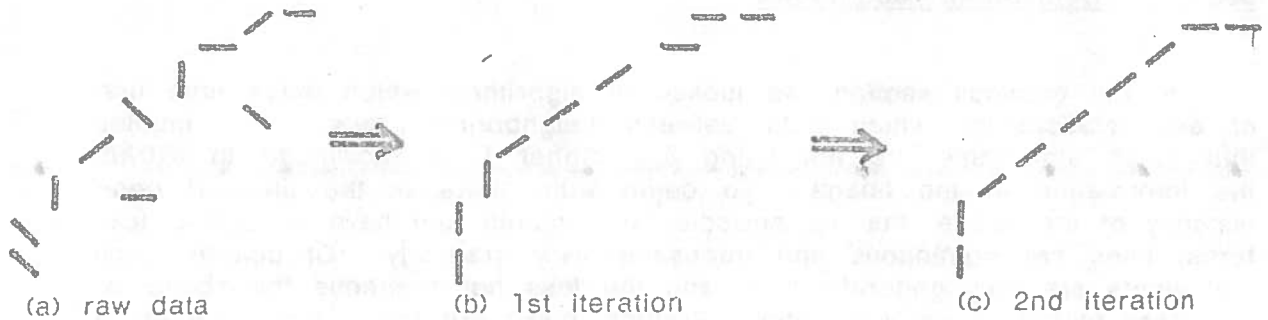


Figure 9 - Hypothetical Example Of A Relaxation Process Acting Upon A Set Of Edge Markers

In figure 9, there is a hypothetical example of a relaxation process iteratively adjusting the labeling of edge markers so as to form a consistent labeling. It also removes a few markers which have no local support. The example shows uncharacteristically good behavior, and should not be taken as an example of the actual performance, rather it just illustrates two effects: consistency improvement and noise reduction. The principle involved is:

Principle

Local consistency relations help reduce classification uncertainities arising because of noise or ambiguity.

The formulation of relaxation algorithms can be expressed in precise mathematical terms (Ull79,Fau80a, Fau80b), but for our purposes, we will give a more informal presentation. The model consists of 6 components:

1. A set of elements. These are the objects upon which the algorithm operates. They may be pixels, regions, line segments or others.
2. A set of possible labels for each element. These are the potential interpretations of the data elements (such as edge orientations, region types, line identifications, vertex types, etc). Each element may receive just one label, or it may have a probability distribution over all possible labels.
3. A neighborhood function. This selects the set of elements which are considered neighbors to each particular element. Examples of neighborhoods are adjacent pixels, or all edges meeting at a vertex (in an image or graph).
4. A neighbor label compatibility function. This is what encodes the higher level knowledge about what is a consistent labeling between neighbors. It assigns some value (say between 0 and 1) to the probability (loosely speaking) that the two (or more) neighboring labels are consistent.
5. An update function. This decides what new label or probability weighting to assign to the current element, based upon its own and its neighbors' labels. This function can be heuristically defined or be given an abstract form based upon elementary probability (Pel80).
6. A control algorithm. Here, we choose to apply this process to all elements in the scene (pixels) in parallel, though sequential processing is also possible. This is an instance of an algorithm whose benefits do not depend upon sequential or parallel

application. The form of the algorithm, however, makes it eminently suitable for parallel execution. The control algorithm has also to decide when to stop the iterations, which is an unsolved problem.

This technique has had a number of different applications, using both low and high level constructs. Work at MIT has tended to use the term "cooperative" to describe their algorithms, but their formulation is very similar to the relaxation formulation (Woo77,Mar76). Further, the global optimization methods (Fau80a,Fau80b) are also very similar. Some examples of applications are:

edge orientation (Zuc77) - as in figure 9
region classification (Han78) - bind classified pixels into more homogeneous regions
cooperative stereo (Mar76,Bar80) - use depth consistency and object continuity to analyze image pairs
cooperative surface orientation (Woo77) - use surface models and image intensity to calculate surface orientations
noise cleaning(Ros78) - image smoothing via local context
line aggregation (Han78) - join edge elements to form lines. (use of continuity and smoothness)
line thinning (Ros78) - reduce thickness of lines found via use of edge detection operators
line labeling (Wal76) - reduce set of possible labels for lines to a (hopefully unique) consistent labeling. (Doesn't use probabilistic notions.)
scene analysis (Hin76) - label image components using required structural relationships.
histogram modification (Ros78) - peak/contrast enhancement
parallel line detection (Dan80) - enhancement of structures with parallel lines (like roads)
curve segmentation (Dav77) - segments the border of a shape into piecewise linear segments
image matching (Ran80) - does pointwise matching of two images (for registration and disparity problems)
relational structure matching (Kit79) - matches one relational structure (pattern) to another (data)

6. Object Based Parallel Algorithms

In section 5, we considered algorithms which were based upon the structural parallelism of the image and its representation. In this section, we base the parallelism upon the contents of the image, or rather the current representation of the image and its contents. It is these distinctions which shift the viewpoint from image processing to computer vision.

We look at three sets of techniques. Firstly, we look at hierarchical techniques, some of which are purely process based and others of which also have data dependencies. Secondly, we look briefly at syntactic techniques, and their parallelism potentials. The final section is somewhat of a catch-all for a few remaining techniques, and some general computer science and artificial intelligence comments.

A general comment should precede the discussion. The algorithms discussed in section 5 were well suited for execution on specialized computer system architectures. This is because the two dimensional structure of the image can be easily mapped onto the array structured machines discussed in section 4A. The algorithms considered here are not (given the present state of the algorithms and machines), and the degree to which the parallelism can actually be realized mainly depends upon the multiprogramming capabilities of the underlying computer system.

6A. Hierarchical Software Structures

In figure 1 (page 9), example 1 (page 10) and the example in figure 6 (page 23) there is embodied a processing hierarchy (explicitly in the first, implicitly in the latter two). Each process, such as the line finder of figure 1, depends upon the results from lower level processes (the edge finder) and passes results up to higher level processes (lines to the vertex finder). Each level produces results with a higher level of conceptual abstraction.

It is also possible to have information flow downward in the hierarchy. This downward flow could be directions for examining the data for hypothesis verification. For example, if the line finder were considering bridging a gap between two segments, then it could direct the edge finder to make a more careful examination of the gap region.

Because of the relatively independent nature of the different levels in the hierarchy, one could structure them to execute in parallel. This is especially true if one process need not wait for the completion of its subordinate processes before it begins.

These concepts lead to:

Principle

> Organize the software about parallelisms inherent in the process, with the hierarchical software structure mirroring the hierarchical conceptual structure.

There are two methodologies considered here. The first is based upon the notion of cooperating experts (specialists in some topic, such as edge finding), organized to contribute to solving the total problem (Les79). This has a strong artificial intelligence character. The second approach is based upon both structuring the processing and reducing the quantity of data.

In figure 10, there is a set of cooperating experts organized to segment images. Each of the experts is a separate process specializing in a particular aspect (color, texture, ...) of the images being analyzed. For example, the boundary detection expert locates what it considers to be object boundaries, based upon color and intensity transitions. It may also include higher level knowledge about the objects expected to be in the scene. Each expert may be decomposable into sub-experts.

No systems based upon a wide variety of knowledge sources such as these have yet been built. The VISIONS (Han78) system analyses the image into boundaries and edges via two independent processes. It uses color to support the region analysis and intensity gradients to support the boundary analysis.

The other class of techniques is based upon the processing cone or pyramid metaphor. The analysis of the image proceeds via the raw image entering at the base of the cone, with reductions in image size and a corresponding increase in data abstraction occurring at each level. The maximally compressed data is available at the apex of the cone. Uhr (Uhr78) gives descriptions of these hierarchical structures, and Tanimoto (Tan78b) gives an overview of the topic as a whole.

A cone consists of a collection of layers, each of which does one

type of processing. A layer consists of a planar array of processing cells. At each level, the cells contain both image based and symbolic data. Between layers of a cone, there is typically a reduction of the amount of the data being processed, say, reducing each 2*2 window to a 1*1. Usually, the only data a processor (a cell in one of the layers) can access is that from its neighbors in the current layer and direct superordinate and subordinate cells. The "recognition cones" of Uhr (Uhr78) have an upward flow of data, while the "processing cones" of Hanson and Riseman (Han78) consider upward ("reduction"), downward ("projection") and horizontal ("iterative") data flow.

This structure has potential for implementation as a hierarchical processor array (Uhr78), but no experimentation has been done.

The structure of a recognition cone developed by Uhr (Uhr78) is given in figure 11. The task of the cone is to segment a natural image containing houses, trees, sky and ground into its component regions. A ten layer cone is chosen for this task, with different processes at most of the ten layers. The results in the tenth layer are very reduced, and suitable only for the most general subsequent processing. (The image has been reduced to a 4*4 array, and about all the information it contains now is that the image contains some identified object, in roughly the designated positions.)

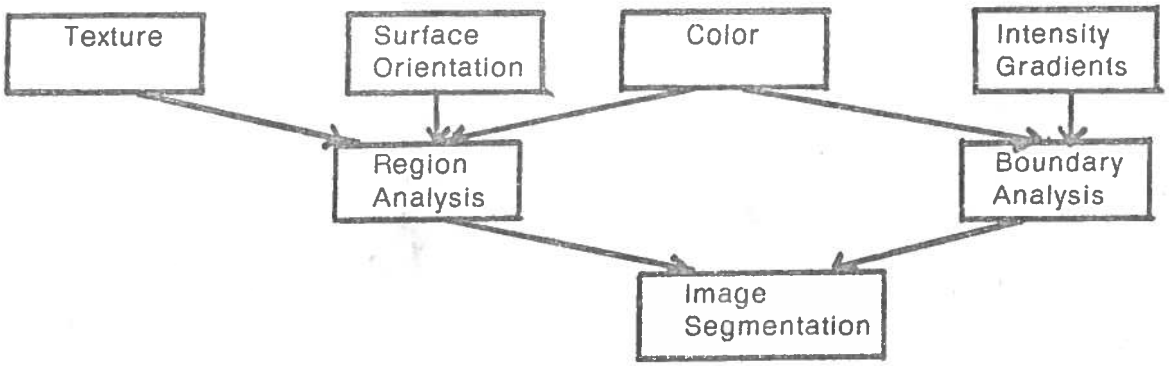


Figure 10 - Hypothetical System Of Cooperating Expert Processes Doing Low Level Image Segmentation
(Each module performs one type of analysis, with its results being used in subsequent analyses.)

Layer	Size	Function
1	800*600	average each of the 3 colors
2	200*200	hue, saturation & intensity.
3	120*120	gradient calculation
4	60*60	short edge detection, simple texture analysis
5	30*30	long edges, angles, curves, textures
6	30*30	compounding & identifications via combinations of regions, borders, angles and textures
7	30*30	more compounding via pattern matching
8	15*15	average labels over 2*2
9	8*8	average labels over 2*2
10	4*4	average labels over 2*2

Figure 11 - A 10 Layer Recognition Cone (Uhr78)

6B. Syntactic Methods

In section 6A, we looked at algorithms which were based upon the parallelisms present in the data and processing structures. In this section, we consider an approach based upon the parallelism between the symbolic elements of the image itself. The discussion consists of a brief overview of the syntactic methods considered, a discussion of how parallelism can be applied to these methods, and lastly an example.

Real world objects have many obvious consistencies. At the low levels, there is color, surface continuity, depth and texture consistency. At higher levels, there is a structural or conceptual consistency: most letters 'b' look roughly similar; most chairs have 4 legs, a seat and a back. It is this consistency which allows successful pattern based approaches to be applied to computer vision.

The syntactic methods discussed here are examples of a pattern based approach, which use the structural properties (consistencies) that exist between the primitive elements of the image (however defined). Some typical primitive elements might be short line segments, small patches of color or texture, or segments with a particular shape (as in the chromosome analysis example discussed below). Assuming the primitive elements of the image have been classified, the syntactic approach consists of iteratively matching groups of elements according to given patterns, and then performing whatever action is required upon a successful match. The actions can include replacing the set of elements by a new element, relabeling the elements, or sending a message (somewhere) that a particular pattern has been found. The description "syntactic methods" applies to this technique, because the matching operations are concerned with the symbolic classifications of the elements, rather than the elements themselves.

The more general form of the pattern matching methods considered here is the type known as production systems (Dav75,Uhr79,Oht79). A production system consists of a set of rules of the form "LHS -> RHS" and a control program. The execution of a rule is such that, whenever the pattern given in the left hand side (LHS) of a rule matches some structure in the database, then the action specified in the right hand side of the rule (RHS) is considered for execution.

More restricted approaches are the grammar based methods. Here, the elements to be matched on the LHS of a rule not only have to be of the correct type, but must also have a specified (in the rule) structural relationship. The best examples of this technique come from studies on natural and programming languages. There are extensive theories on the structure and properties of grammars, the languages they describe, and the procedures for effectively parsing sentences of the languages (Aho72).

Most grammar research has concentrated upon linear strings of symbols, such as one encounters in parsing sentences of natural languages. Images, on the other hand, have a two dimensional structure. As a result, special methods have been considered to augment the one-dimensional notation. These include:

web grammars (Ros72) - Refer to figure 12. These are sets of rules which direct how to make parses in a graph-like structure. The graph structure is useful for representing the many relationships that can exist between the elements of an image. The nodes of

the graph can be relations like "adjacent" or elements like "piece of white surface" or "chair leg".

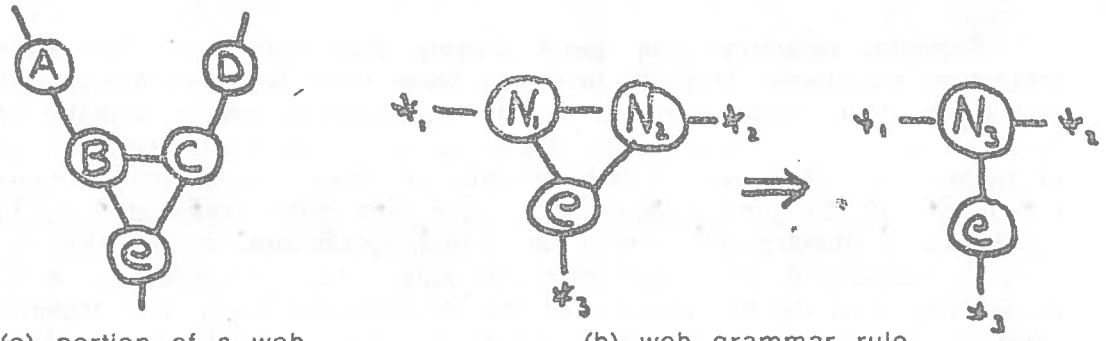
tree grammars (Lu78,Moa76) - Refer to figure 13. These are sets of rules whose terminal nodes are restricted to have a particular two dimensional structure, such as that of figure 13a.

description languages (Mil68) - The two dimensional structure is described by a one dimensional string, which is then parsable by more standard methods. The chromosome analysis example given below is an example of this technique. A good overview of early work on such techniques is given by Miller (Mil68).

plex grammars (Fu74) - Each of the symbols (terminal & non-terminal) is augmented with a set of attaching points. Rules of grammar include descriptions of where the elements are attached. This technique has been useful for parsing line based structures, such as electrical circuits, flowcharts and chemical structures.

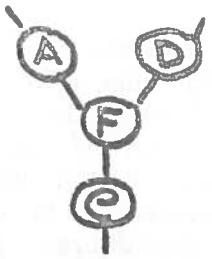
mosaic grammars (Ota75) - Grammars based upon regular tilings of the image. Rules describe allowable relationships between the tiles.

These methods are most evidently syntactic, in that the grammars used for the parsing of the image are given explicitly. K.S. Fu gives a good presentation of the subject in his book (Fu74).



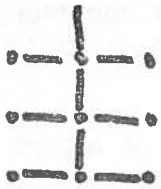
(a) portion of a web (nodes represent derived symbolic structures)

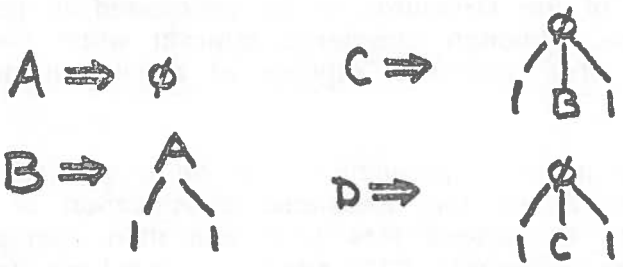
(b) web grammar rule (a rule matching terminal, nonterminal and unspecified nodes in a specific configuration)



(c) reduced web (produced from (a) by use of rule in (b))

Figure 12 - A Simple Web Grammar Example





(a) tree structure (shape of structure to be parsed)

(b) grammar rules (matching on pixel values and parsed nodes having the given structure)

(c) tree parsable by grammar (using the structure from (a) and rules from (b))

Figure 13 - A Simple Tree Grammar Example

Previous research has been largely concerned with the parsing of artifactual structures. Images, however, have their basis in natural structure, and hence have large amounts of both randomness and essentially irrelevant detail. Further, the process of acquiring an image introduces the problems of noise. To cope with these aspects of images (and other natural systems), the notion of stochastic grammars has been invented (Lee72,Swa72). Stochastic grammars are similar to normal grammars, except that a probability is associated with each grammar rule. The probability of a particular parse may then be the product of the probabilities of all the grammar rules used in its derivation (or some other function). Additional grammar rules are introduced to account for the less likely structures that occur in the base language (in our case, the representation of the image). These additional rules are likely to lead to ambiguous parses, from which the parser selects the parsing(s) with the highest probability.

We now consider where parallelism fits into this discussion. The major feature of both the production rule and grammar rule approaches is the number of rules. In realistic cases, the number can be quite large – say on the order of 200–1000. Each of these rules is independent in terms of its initiation and execution (though the effects of one rule often cause another rule to become enabled). In essence, each of the rules represents a parallel independent procedure. So, a reasonable computational method for such a set of procedures is to allow each of them to execute in parallel (Uhr79).

In the case of images, there is likely to be a large number of similar structures, which can lead to another mode of parallelism. In that all of the structures in the image are generally independent, then it is reasonable to allow all of the structures to be processed in parallel. This would work well, because, although structures interact when they are parsed together, the number of independent regions of activity in an image is likely to be high.

Another mode of parallelism can be considered. In the work of Moayer and Fu (Moa76) on the automatic classification of fingerprints, they divide the print into 16 regions (4*4 grid) and then attempt to parse each region. They use approximately 200 different grammars for parsing the regions, among which only one is expected to successfully parse any given region. The identifications of the grammars which parse the print are then used to form its classification. Thus it is possible to also execute the 190 parsers, in parallel, over the 16 regions, in parallel.

It would appear as if the notions of parallel rule sets, parallel rules and parallel data complement each other well. We summarize these considerations as:

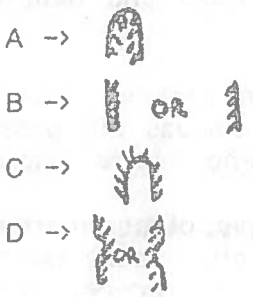
Principle

> Consider consistency in image data and structures as conceptual units upon which to build processing rules (syntactic or procedural). Take advantage of the independence of the rules and data units to execute in parallel.

There has not been much research on parallel parsing in the image processing context. Chang and Fu (Cha79) discuss the possible applications to tree grammar parsings of LANDSAT data and texture discrimination.

We conclude this section with an example of chromosome analysis discussed by Fu (Fu74). Chromosome images are initially two dimensional, but prior to parsing, an image processing routine reduces the image of the chromosome to a one dimensional encoding of its boundary properties. In figure 14, we list the terminal symbols and a portion of the chromosome grammar. In figure 15, we give a hypothetical chromosome (labelled by its terminal symbols) and the associated parse tree.

In terms of our parallelism considerations, we can achieve at least a factor of 3 improvement here. Given the parse tree in figure 15, the symbols at the greatest depth can be parsed at the first cycle, the next greatest at the next cycle, and so on. This tree has 6 levels and 21 nodes, hence roughly 3 nodes parsed per cycle. Of course, this doesn't include any of the parsings which didn't contribute to the final tree. If we were parsing a realistic two dimensional structure, then the opportunities for parallelism would undoubtedly have been considerably greater.

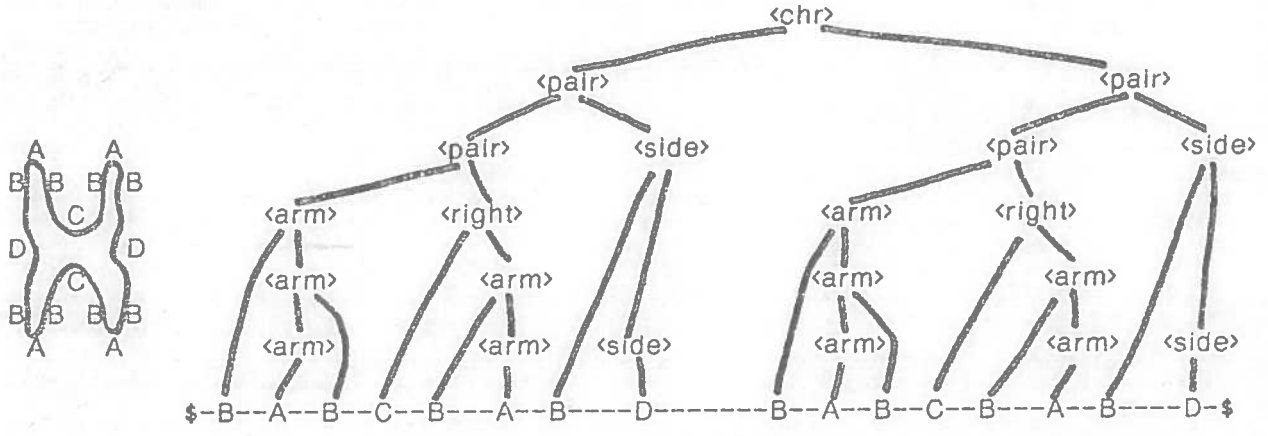


(a) terminal symbol mapping (bits of edge of chromosome)

<chr> -> <pair><pair>
<pair> -> <pair><side>
<pair> -> <arm><right>
<side> -> B<side>
<side> -> D
<right> -> C<arm>
<arm> -> B<arm>
<arm> -> <arm>B
<arm> -> A

(b) grammar rules

Figure 14 - Chromosome Analysis Grammar



(a) terminal labelled chromosome

(b) parse tree for chromosome (from chromosome terminal labels arranged into linear string)

Figure 15 - Example Of Chromosome Parsing

6C. Other Software Techniques

This section is intended as a place to mention a few other techniques not appropriately categorized elsewhere. It includes both vision specific and general computer science/artificial intelligence techniques.

There are currently a few languages, designed for image analysis, which actually attempt to represent the parallel structure of the problem. The first category of these languages include L (Rad81), PascalPL (Uhr81) and PIXAL (Lev81). These languages are normal algorithmic languages with extensions to have declared image array types, and operations which execute in parallel over the arrays. The operations include arithmetic, logical and conditional functions. The second category views the problem not as a single process over a parallel data structure, but as a parallel collection of processes over a single data structure, executing cooperatively. The language MAC (Dou81) exemplifies this view, and has constructs for two dimensional arrays of processes. Here, one can associate a process with each pixel in an array. It also includes the control constructs of modern concurrent languages.

Mero (Mer78) describes a line finding and vertex connection algorithm. It is based upon parallel searching of the image along potential edge cells. Merlin and Farber (Mer75) describe a parallel implementation of the Hough transform, as applied to curve detection.

There are also some general artificial intelligence techniques suitable for parallel vision applications. The first technique is that of search (Nil80). If the analysis of the image requires consideration of alternative interpretations or actions, and a subsequent decision to pursue one of the alternatives and temporarily skip the others, then we have a framework for search techniques. In this case, a separate processor can be assigned to pursue each alternative in parallel. The next technique is the use of ACTOR networks (Hew77). This structure is a collection of independent processes (actors), each dedicated to performing a specific task. While processing the request, the actor may send messages to other actors. One advantage of this technique is the ability to define a prototype actor (such as LINE_FOLLOWER) and then create multiple instances of the actor as needed.

There is also the possibility of the use of perceptron-like processes (Nil65). These are essentially linear discriminant functions, (also called threshold logic units) and a large collection of simple units may be easily connected into a substantial structure (hierarchical or planer). Because of their simple computational structure, they can easily be implemented in hardware, especially in arrays using VLSI techniques. However, it is not clear how they might be used for more than very low level tasks (such as edge detection and line joining). They have also been shown to be incapable of performing some processes, such as global recognition of connectivity (Min69).

7. Summary Of Techniques

In this section, we summarize the potential benefits associated with each of the techniques discussed previously. At the end of the section there are several examples of system architectures which perform the inspection task of example 1 at reasonable speeds.

This paper examined and discussed hardware based techniques. The major ones, and their purposes are:

optical - use optical Fourier transform techniques to do complete image analysis simultaneously.
array structures - build a processor array corresponding to the image array so as to execute local operations in parallel.
pipelines - develop custom logic units to process the data upon receipt and then pass it on to the next unit. Process the separate stages in parallel.
functional units - develop custom logic units as auxiliaries to main processors to do specialized processing at high speeds. Execute the auxiliary units in parallel.
general parallel processors - execute conventional programs in parallel.

The paper also examined and discussed software techniques used to give the image processing task a parallel structure. The major techniques and their purposes are:

parallel array algorithms - execute the algorithm at each local neighborhood in an image. All executions take place simultaneously.
parallel relaxation - same as above, but also add image consistency knowledge.
processing pipelines - decompose process into sequential subprocesses. Execute subprocesses in parallel.
high level hierarchies - decompose process into independent contributing experts. Execute independent modules in parallel.
processing cones - structure process into levels of data compression and abstraction. Also uses parallel array methods.
syntactic methods - execute pattern based processing in parallel.
general parallel programming - execute independent code in parallel.

In table 4, there is a summary of the estimated processing improvement factors. Note that there are factors due to both parallelism and special architectures. For the low level (image data based) processing there is potential for a 10^3 - 10^4 improvement in performance over conventional processors. For the higher level processing (object based), there appears to be only about a 10^1 - 10^2 improvement given current techniques. However, as a majority of current processing lies at the low level, both figures are reasonably suited to the processing required, as shown in figure 1.

For the remainder of this section, we consider the design of an architecture which performs the inspection process given in example 1. (This example does a greyscale inspection of a manufactured object.)

We use two sets of requirements, one to process the images in real time (with video rate input, one frame every 25 milliseconds), and the other to process one image every 1 second. Hypothetical system architectures for the two case are shown in figures 16 and 17. Tables 5 and 6 give

summaries of the steps needed in the processing, the techniques used for each step, the delays estimated inherent in the processing, and the processor utilizations.

We consider the configuration in figure 16. If we use pipeline processors for the initial image processing, sixteen way parallelism between normal microcomputers for line finding (with 16 image buffers) and two future minicomputers with built-in 10 way parallelism, then we anticipate achieving real time processing rates. This would allow processing each picture as received, with a 445 millisecond delay. The estimated cost for such a system is on the order of $90,000, with 1985 as a feasible construction date.

If there were less of a speed requirement, and we could accept images one second apart, then the system from figure 17 is usable. The pipeline stages are still used, but all of the subsequent processing stages are now done in a microcomputer system with an attached frame store. In this case, the estimated delay is 705 milliseconds per image. The estimated cost of this processing system is $7,000, with 1983 as a feasible construction date. This is a very reasonable price for this system.

Technique	Parallelism	Other factors
HARDWARE		
optical	10^6	10^-5 (image introduction)
array structures	10^4	
pipeline units	10	10^2 (custom architectures)
functional units	5	10^1 to 10^2 (custom architectures)
general parallel processing	10	
SOFTWARE (requires implementation on one of the above architectures)		
parallel arrays	10^4	
parallel relaxation	10^4	
processing pipeline	10	
high level hierarchy	5	
processing cone	10^3	
syntactic methods.	10^1 to 10^2	
general parallel processing	10	

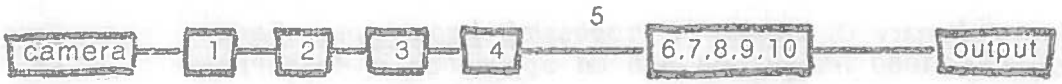
Table 4 - Summary Of Estimated Processing Improvement Factors
(assumes 1000*1000 image and optimal application of techniques)

pipeline processors (1983 - $200 ea)

16 microcomputers and associated framestore (1982 - $4,000 ea)

2 10-parallel minicomputers (1985 - $10,000 ea)

Figure 16 - Hypothetical System Architecture For Inspection Task At 1 Frame Every 25 Milliseconds
(numbers in boxes refer to step numbers in Table 5)



pipeline processors (1983 - $200 ea)

microcomputer and associated frame store (1982 - $4,000)

Figure 17 - Hypothetical System Architecture For Inspection Task At 1 Frame Every 1 Second
(numbers in boxes refer to step numbers in Table 6)

Processing Delay Times

Step	Action	Hardware	Algorithm	Incremental Delay (msec)
1	preprocessing convolution	pipeline	-	2
2	edge mask	pipeline	-	3
3	threshold	pipeline	-	0
4	edge thinning	pipeline	-	5
5	image acquisition	-	-	25
6	edge tracking	microcomputer	serial	50
7	line finding	microcomputer	serial	200
8	line parameter	microcomputer	serial	20
9	parameter transmission	-	-	100
10	line joining/vertex location	10-parallel	parallel search	20
11	model matching	10-parallel	syntactic	20
				445 (total)

Processor Utilizations (assuming 1 frame every 25 milliseconds)

Processor	Processing (msec)	Interval (msec)	Utilization (%)
1	25	25	100
2	25	25	100
3	25	25	100
4	25	25	100
6,7,8 + 9	370	400	93
10	20	25	80
11	20	25	80

Table 5 - Processing Time Analysis Of Example 1 Task When Executed On Architecture Of Figure 16

Processing Delay Times

Step	Action	Hardware	Algorithm	Incremental Delay (msec)
1	preprocessing convolution	pipeline	-	2
2	edge mask	pipeline	-	3
3	threshold	pipeline	-	0
4	edge thinning	pipeline	-	5
5	image acquisition	-	-	25
6	edge tracking	microcomputer	serial	50
7	line finding	microcomputer	serial	200
8	line parameter	microcomputer	serial	20
9	line joining/vertex location	microcomputer	serial	200
10	model matching	microcomputer	syntactic	200
				705 (total)

Processor Utilizations (assuming 1 frame every 1 second)

Processor	Processing (msec)	Interval (msec)	Utilization (%)
1	25	1000	3
2	25	1000	3
3	25	1000	3
4	25	1000	3
6-10	670	1000	67

Table 6 - Processing Time Analysis Of Example 1 Task When Executed On Architecture Of Figure 17

8.Summary

Considering all of the approaches under research, as surveyed in this paper, there are plenty of opportunities for applying parallelism to the processing of images. These opportunities are present at both the low and high levels of image analysis, and, considering the amount of processing power needed for the task, the research is both relevant and timely.

In this paper, we looked at optical techniques that have been previously used (section 3) and some of the computer architecture innovations that form a base for parallelism (section 4). These include array structures (4A), pipeline structures (4B) and special asynchronous functional units (4C). In section 5, we discussed the data structure based software techniques of array algorithms (5A) and relaxation techniques (5B). In section 6, we discussed the more conceptually based hierarchical techniques (6A) and syntactic techniques (6B).

As a preliminary to the techniques, we motivated the discussion with an analysis of the need for parallelism in image processing and understanding. As part of the concluding discussion, we included a summary and comparison of the improvement factors potentially achievable using current or shortly available technology. Lastly, we gave brief descriptions of two hypothetical systems which, in the near future, might achieve almost real time image processing for a reasonable price.

Although all of the techniques discussed in the paper offer improved system performance, only a few are likely to be of significant value (author's opinion). The preferred hardware technologies are those of the parallel arrays and specialized pipeline processors (though the major benefits of the pipeline units come from their high processing rates, instead of the marginal parallelism that they support.) Among the software techniques, the parallel array algorithms offers promise to the lower levels of processing (where, at present, most processing is required). For the higher levels, the parallel syntactic analysis seems to be a potential alternative to current sequential processing methods (though the use of this technology is not well developed). Lastly, the development of general purpose processors, which execute conventional programs in parallel, offers increased general performance.

9.References

[Aho72] Aho,A.V., Ullman,J.D., The Theory Of Parsing, Translation, And Compiling, Vol 1: Parsing, Prentice-Hall, 1972

[ano81] anonymous, **Image Processing Group Research Review**, Dept. Of Physics and Astronomy, University College Of London, 1981

[Arc75] Arcelli,C., Cordella,L., Levialdi,S., **"Parallel Thinning Of Binary Pictures"**, Electronics Letters, Vol 11, Apr 1975

[Baj73] Bajcsy,R., **"Computer Identification Of Textured Visual Scenes"**, PhD thesis, Stanford University, 1973

[Bar68] Barnes,G.H., Brown,R.M., Kato,M., Kuck,D.J., Slotnick,D.L., Stokes,R.A., **"The ILLIAC IV Computer"**, IEEE Trans. Computers, Vol C-17, #8, 1968, pp 746-757

[Bar80] Barnard,S., Thompson,W., **"Disparity Analysis Of Images"**, IEEE Trans. Pattern Analysis & Machine Intell., Vol PAMI-2, #4, 1980

[Bas81] Basille,J.L., Castan,S., Latil,J.Y., **"Systeme Multiprocesseur Adapte au Traitement d'Images"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp205-214, 1981.

[Bat80] Batcher,E., **"Design Of A Massively Parallel Processor"**, IEEE Trans. Comp. Vol C-29, #9, 1980, pg836

[Bla81] Blake,A., **"Fixed Point Solutions Of Recursive Operations On Boolean Arrays"**, Edinburgh Machine Intelligence Research Report MIP-R-131, 1981

[Bra65] Bracewell,R., The Fourier Transform And Its Applications, McGraw-Hill, 1965.

[Bra78] Brabston,D.C., Traber,J.E., **"Design Of Pipeline Systems For Landsat Image Processing"**, Proc. AFIPS 1978 NCC, Vol 47, pg 151

[Bri79] Briggs,F.A., Fu,K.S., Hwang,K., Patel,J., **"PM^4 - A Reconfigurable Multiprocessor System For Pattern Recognition And Image Processing"**, Proc. AFIPS 1979 NCC, Vol 48, pg 255

[Cas75] Casasent,D., Sterling,W., **"An Optical/Digital Processor: Hardware And Applications"**, IEEE Trans. Comp., Vol C-24, #4, 1975, pg 348

[Cas78] Casasent,D. (editor), Optical Data Processing, Applications, Springer-Verlag, 1978

[Cha79] Chang,N.S., Fu,K.S., **"Parallel Parsing Of Tree Languages For Syntactic Pattern Recognition"**, Pattern Recognition, Vol 11, 1979, pg 213

[Coo77] Cooper,R.G., **"The Distributed Pipeline"**, IEEE Trans. Comp., Vol C-26, #11, 1977, pg 1123

[Cor78] Cordella,L.P., Duff,M.J.B., Levialdi,S., **"An Analysis Of Computational Cost In Image Processing: A Case Study"**, IEEE Trans. Comp. Vol C-27, #10, 1978, pg 904

[Dan80] Danker,A., Rosenfeld,A., **"Strip Detection Using Relaxation"**, Pattern Recognition, Vol 12, 1980, pg 269

[Dav77] Davis,L.S., Rosenfeld,A., **"Curve Segmentation By Relaxation Labeling"**, IEEE Trans. Comp. Vol C-26, #10, 1977, pg 1053

[Dav75] Davis,R., King,J., **"An Overview Of Production Systems"**, Stanford AI Lab memo 271, 1975

[Dou81] Douglas,R.J., **"MAC: A Programming Language for Asynchronous Image Processing"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp41-52, 1981.

[Duf73] Duff,M.J.B., Watson,D.M., Fountain,T.J., Shaw,G.K., **"A Cellular Logic Array For Image Processing"**, Pattern Recognition, Vol 5, 1973, pg 229

[Duf76] Duff,M.J.B., **"CLIP-4: A Large Scale Integrated Circuit Array Parallel Processor"**, 3rd IJCPR, 1976, pg 728

[Duf77] Duff,M.J.B., WatsonDM, **"The Cellular Logic Image Processor"**, Computer Journal, Vol 20, 1977, pg 68

[Duf78] Duff,M.J.B., **"Review Of The CLIP Image Processing System"**, Proc AFIPS 1978 NCC, pp 1055-1060

[Fau80a] Faugeras,O., **"An Optimization Approach For Using Contextual Information In Computer Vision"**, Proc 1980 National Conf. On AI, 1980, pg 56.

[Fau80b] Faugeras,O., Berthod,M., **"Scene Labeling: An Optimization Approach"**, Pattern Recognition, Vol 12, 1980, pg 339

[Fin77] Finnila,C.A., Love,H.H., **"The Associative Linear Array Processor"**, IEEE Trans Comp. Vol C-26, #2, 1977, pg 112

[Fu74] Fu,K.S., Syntactic Methods In Pattern Recognition, Academic Press, 1974.

[Fu78a] Fu,K.S., **"Special Computer Architectures For Pattern Recognition And Image Processing - An Overview"**, Proc. AFIPS 1978 NCC, Vol 47 pg 1003

Processing - An Overview", Proc. AFIPS 1978 NCC, Vol 47 pg 1003

[Gem81] Gemmer,P., Ischen,H., Luetjen,K., **"FLIP: A Multiprocessor for Image Processing"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp245-256, 1981.

[Ger81] Gerritsen,F.A., Monhemius,R.D., **"Evaluation of the Delft Image Processor DIP-1"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp189-204, 1981.

[Gol69] Golay,M.J.E., **"Hexagonal Parallel Pattern Transforms"**, IEEE Trans. Comp. Vol C-18, #8, 1969, pg 733

[Gra81] Granlund,G.H., **"GOP: A Fast and Flexible Processor for Image Analysis"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp 179-188, 1981.

[Han78] Hanson,A.R., Riseman,E.M., **"Segmentation Of Natural Scenes"**, in Computer Vision Systems, eds. Hanson and Riseman, Academic Press, 1978, pg 129

[Hew77] Hewitt,C., **"Viewing Control Structure As Patterns Of Passing Messages"**, Artificial Intelligence, Vol 8, 1977, pp 323-364

[Hin76] Hinton,G., **"Using Relaxation To Find A Puppet"**, Proc. 1976 Artif. Intel. And Simul. Of Behaviour Conference, pg 148

[Hub63] Hubel,D.H., **"The Visual Cortex Of The Brain"**, Scientific American, Vol 209, #5, 1963, pp 54-62

[Hum78] Hummel,R.A., Rosenfeld,A., **"Relaxation Processes For Scene Labeling"**, IEEE Trans. Sys, Man, and Cyber, Vol SMC-8, #10, 1978, pg 765

[Hus73] Husain-Abidi,A.S., **"Design Concepts For An On-Board Parallel Image Processor"**, Pattern Recognition, Vol 5, 1973, pg 13

[Jac73] Jacobson,A.D., Beard,T.D., Bleha,W.P., Maugerum,J.D., Wong,S.Y., **"The Liquid Crystal Light Valve, An Optical-To-Optical Interface Device"**, Pattern Recognition, Vol 5, 1973, pg 13

[Kit79] Kitchen,L., Rosenfeld,A., **"Discrete Relaxation For Matching Relational Structures"**, IEEE Trans. Sys., Man, and Cyber., Vol SMC-9, #12, 1979, pg 869

[Koc75] Kock,W., **"A Real-Time Parallel Optical Processing Technique"**, IEEE Trans. Comp. Vol C-24, #4, 1975, pg 407

[Kru73] Kruse,B., **"A Parallel Picture Processing Machine"**, IEEE Trans. Computers, Vol C-22, #12, Dec 1973, pg 1075

[Kru78] Kruse,B., **"Experiments With A Picture Processor In Pattern Recognition Processing"**, Proc. AFIPS 1978 NCC, Vol 47, pp 1015-1024

[Kun80] Kung,H.T., **"Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI"**, Carnegie-Mellon report CMU-CS-80-132, July 1980.

[Lan79] Landgrebe,D., **"Monitoring The Earth's Resources From Space - Can You Really Identify Crops By Satellite?"**, Proc. AFIPS 1979 NCC, Vol. 48, pp 233-241

[Lan81] Lantuejoul,C., **"An Image Analyser"**, in Duff & Leviaidi (eds), "Languages and Architectures for Image Processing", pp165-178, 1981.

[Lee72] Lee,H.C., Fu,K.S., **"A Stochastic Syntax Analysis And Its Application To Pattern Classification"**, IEEE Trans. Comp. Vol C-21, #7, 1972, pg 660

[Lee73] Lee,S.H., **"The Synthesis Of Complex Spatial Filters For Coherent Optical Data Processing"**, Pattern Recognition, Vol 5, 1973, pg 21

[Lev81] Levialdi,S., Maggiolo-Schettini,A., Napoli,M., Tortora,G., Uccella,G., **"On the Design and Implementation of PIXAL, a Language for Image Processing"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp89-98, 1981.

[Lin72] Lindsey,P.H., Norman,D.A., **"Human Information Processing"**, Academic Press, 1972

[Lov80] Love,H.H., **"The Highly-Parallel Supercomputers: definitions, applications and predictions"**, Proc. AFIPS 1980 NCC, Vol 49, pg 181

[Lu78] Lu,S.Y., Fu,K.S., **"Error Correcting Tree Automata For Syntactic Pattern Recognition"**, IEEE Trans. Comp. Vol C-27, #11, 1978, pg 1040

[Man81] Manera,R., Stringa,L., **"The EMMA System: An Industrial Experience on a Multiprocessor"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp215-228, 1981.

[Mar74] Marr,D.,**"An Essay On The Primate Retina"**, MIT AI memo 296, 1974

[Mar76] Marr,D., Poggio,T., **"Cooperative Computation Of Stereo Disparity"**, MIT AI memo 364, 1976

[Mar77] Marr,D., **"Representing Visual Information"**, MIT AI Memo, AIM-415, 1977

[Mar80] Marks,P., **"Low Level Vision Using An Array Processor"**, Comp. Graphics and Image Proc., Vol 14, #3, 1980, pg281

[Mat79] Matsushima,H., Uno,T., Ejiri,M., **"Image Processing By Experimental Arrayed Processor"**,

Proc. 6th IJCAI, 1979, pp S13-S15

[Mer75] Merlin,P.M., Farber,D.J., **"A Parallel Mechanism For Detecting Curves In Pictures"**, IEEE Trans. Comp., Vol C-24, 1975, pg 96

[Mer78] Mero,L., **"A Quasi-Parallel Contour Following Algorithm"**, Proc. 1978 Artif. Intel. And Simul. Of Behaviour Conf., pg 189

[Mil68] Miller,W.F., Shaw,A.C., **"Linguistic Methods In Picture Processing - A Survey"**, Proc. AFIPS 1968 FJCC, pg 279

[Min69] Minsky,M., Papert,S., Perceptrons, An Introduction To Computational Geometry, MIT Press, 1969

[Moa76] Moayer,B., Fu,K.S., **"A Tree System Approach For Fingerprint Pattern Analysis"**, IEEE Trans. Comp. Vol C-25, #3, 1976, pg 262

[Mor78] Mori,K.I., Kidode,M., Shinoda,H., Asada,H., **"Design Of Local Parallel Pattern Processor For Image Processing"**, Proc. AFIPS 1978 NCC, pp 1025-1031

[Nil65] Nilsson,N., Learning Machines, McGraw-Hill, 1965

[Nil80] Nilsson,N., Principles Of Artificial Intelligence, Tioga Publishing, 1980

[Nud80] Nudd,G.R., **"Image Understanding Architectures"**, Proc. AFIPS 1980 NCC, pp 377-390

[Ogo76] O'Gorman,F., **"Edge Detection Using Walsh Functions"**, Proc. 1976 Artif. Intel. And Simul. Of Behaviour Conf., pg 195

[Oht79] Ohta,Y., Kanade,T., Sakai,T., **"A Production System For Region Analysis"**, Proc. 6th IJCAI, 1979

[Ota75] Ota,P.A., **"Mosaic Grammars"**, Pattern Recognition, Vol 7, 1975, pg 61

[Pat80] Patel,J., **"An Alternative To The Distributed Pipeline"**, IEEE Trans. Computers, Vol C-29, #8, 1980, pg 736

[Pel80] Peleg,S., **"A New Probabilistic Relaxation Scheme"**, IEEE Trans. Pattern Analysis And Mach. Intel. Vol PAMI-2, #4, 1980, pg 362

[Pot78] Potter,J.L., **"The STARAN Architecture And Its Application To Image Processing And Pattern Recognition Algorithms"**, Proc. AFIPS 1978 NCC, pg 1041

[Pre71] Preston,K., **"Feature Extraction By Golay Hexagonal Pattern Transforms"**, IEEE Trans. Comp. Vol C-20, #9, 1971, pg 1007

[Pri77] Price,K., Reddy,R., **"Change Detection And Analysis In Multispectral Images"**, Proc. 5th IJCAI, 1977, pp 619-625

[Rad81] Radhakrishnan,T., Barrera,A., Guzman,A., Jinich,A., **"Design of a High Level Language (L) for Image Processing"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp25-40, 1981.

[Ran80] Ranade,S., Rosenfeld,A., **"Point Pattern Matching By Relaxation"**, Pattern Recognition, Vol 12, 1980, pg 269

[Red79] Reddy,D.R., Hon,R.W., **"Computer Architectures For Vision"**, in Computer Vision And Sensor Based Robots, eds Dodd and Rossol, Plenum Press, 1979, pp 169-186

[Ree80a] Reeves,A.P., **"A Systematically Designed Binary Array Processor"**, IEEE Trans. Comp., Vol C-29, #4, 1980, pg278

[Ree80b] Reeves,A.P., **"On Efficient Global Information Extraction Methods For Parallel Processors"**, Comp. Graphics and Image Proc., Vol 14, #2, 1980, pg 159

[Roe78] Roesser,R.P., **"Two-dimensional Microprocessor Pipelines For Image Processing"**, IEEE Trans. Comp. Vol C-27, #2, 1978, pg 144

[Roh77] Rohrbacher,D., Potter,J.L., **"Image Processing With The STARAN Parallel Computer"**, Computer, Vol 10, #8, Aug 1977, pg 54

[Ros72] Rosenfeld,A., Milgram,D.L., **"Web Automata And Web Grammars"**, in Machine Intelligence 7, eds. Meltzer and Michie, Edinburgh University Press, 1972

[Ros78] Rosenfeld,A., **"Iterative Methods In Image Analysis"**, Pattern Recognition, Vol 10, 1978, pg 181

[Sie81a] Siegel,H.J., **"PASM: A Reconfigurable Multi-microprocessor for Image Processing"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp257-266, 1981.

[Sie81b] Siegel,L.J., **"Image Processing on a Partitionable SIMD Machine"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp 293-300, 1981.

[Skl72] Sklansky,J., Nahin,P.J., **"A Parallel Mechanism For Describing Silhouettes"**, IEEE Trans. Comp. Vol C-21, #11, 1972, pg 1233

[Skl76] Sklansky,J., Cordella,L.P., Levialdi,S., **"Parallel Detection Of Concavities In Cellular Blobs"**, IEEE Trans. Comp. Vol C-25, #2, 1976, pg 187

[Sor80] Soroko,L.M., Holography And Coherent Optics, Plenum Press, 1980, (translated from corrected

1971 text)

[Sta75a] Stark,H., **"An Optical-Digital Computer For Parallel Processing Of Images"**, IEEE Trans. Comp. Vol C-24, 1975, pg 340

[Sta75b] Stamopoulos,C., **"Parallel Image Processing"**, IEEE Trans. Comp. Vol C-24, #4, 1975, pg 424

[Swa72] Swain,P.H., Fu,K.S., **"Stochastic Programmed Grammars For Syntactic Pattern Recognition"**, Pattern Recognition, Vol 4, 1972, pg 83

[Tan78a] Tanimoto,S.L., **"Regular Hierarchical Image And Processing Structures In Machine Vision"**, in Computer Vision Systems, eds. Hanson and Riseman, Academic Press,1978,pg 165

[Tan78b] Tanimoto,S.L., **"An Optimal Algorithm For Computing Fourier Texture Descriptors"**, IEEE Trans. Comp. Vol C-27, #1, 1978, pg 81

[Ter55] Terman,F.E., Electronic And Radio Engineering, McGraw-Hill, 1955, ch 25

[Tip68] Tippett,J.T., Berkowitz,D.A., Clapp,L.C., Koester,C.J., VanderburghA (editors), Optical And Electro-optical Information Processing, MIT press, 1965

[Uhr78] Uhr,L.,**"'Recognition Cones', And Some Test Results; The Imminent Arrival Of Well-Structured Parallel-Serial Computers; Positions And Positions On Positions"**, in Computer Vision Systems, eds. Hanson and Riseman, Academic Press, 1978, pg 363

[Uhr79] Uhr,L., **"Parallel-Serial Production Systems"**, Proc. 6th IJCAI, 1979

[Uhr81] Uhr,L., **"A Language for Parallel Processing of Arrays, Embedded in PASCAL"**, in Duff & Levialdi (eds), "Languages and Architectures for Image Processing", pp53-82, 1981.

[Ull79] Ullman,S., **"Relaxation And Constrained Optimization By Local Processes"**, Comp. Graphics And Image Proc., Vol 10, #2, 1979, pp115-125

[Vic78] Vick,C.R., Cornell,J.A., **"PEPE Architecture - Present And Future"**, Proc. AFIPS 1978 NCC, Vol 47, pg 981

[Ver78] Van Voorhis,D.C., Morrin,T.H., **"Memory Systems For Image Processing"**, IEEE Trans. Comp. Vol C-27, #2, 1978, pg 113

[Wal76] Waltz,D., **"Understanding Line Drawings Of Scenes With Shadows"**, in The Psychology Of Computer Vision, ed Winston, McGraw-Hill, 1976, pg 19

[Wal78] Waltz,D., **"A Parallel Model For Low Level Vision"**, in Computer Vision Systems, eds. Hanson and Riseman, 1978, Academic Press, pg 175

[Wat79] Watson,I., Gurd,J., **"A Prototype Data Flow Computer With Token Labeling"**, Proc. AFIPS 1979 NCC

[Woo77] Woodham,R.J., **"A Cooperative Algorithm For Determining Surface Orientation From A Single View"**, Proc. 5th IJCAI, 1977, pg 635

[Zuc77] Zucker,S.W., Hummel,R.A., Rosenfeld,A., **"An Application Of Relaxation Labeling To Line And Curve Enhancement"**, IEEE Trans. Comp. Vol C-26, #4, 1977, pg 394 and corrections Vol C-26, #9, pg 922

[Zuc78a] Zucker,S.W., **"Vertical And Horizontal Processes In Low Level Vision"**, in Computer Vision Systems, eds. Hanson and Riseman, 1978, Academic Press, pg 187

[Zuc78b] Zucker,S.W., Krishnamurthy,E.V., Haar,R.L., **"Relaxation Processes For Scene Labeling: Convergence, Speed And Stability"**, IEEE Trans. Sys, Man and Cyber., Vol SMC-8, #1, 1978, pg 41