PARALLEL IMPLEMENTATION OF IMAGE SEGMENTATION ALGORITHMS ON A NETWORK OF TRANSPUTERS

WATHIQ MOSLEM MANSOOR

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

April 1990

The University of Aston in Birmingham

PARALLEL IMPLEMENTATION OF IMAGE SEGMENTATION ALGORITHMS ON A NETWORK OF TRANSPUTERS

WATHIQ MOSLEM MANSOOR

Doctor of Philosophy

April 1990

Summary

Image segmentation is one of the most computationally intensive operations in image processing and computer vision. This is because a large volume of data is involved and many different features have to be extracted from the image data. This thesis is concerned with the investigation of practical issues related to the implementation of several classes of image segmentation algorithms on parallel architectures. The Transputer is used as the basic building block of hardware architectures and Occam is used as the programming language. The segmentation methods chosen for implementation are convolution, for edge-based segmentation; the Split and Merge algorithm for segmenting non-textured regions; and the Granlund method for segmentation of textured images. Three different convolution methods have been implemented. The direct method of convolution, carried out in the spatial domain, uses the array architecture. The other two methods, based on convolution in the frequency domain, require the use of the two-dimensional Fourier transform. Parallel implementations of two different Fast Fourier Transform algorithms have been developed, incorporating original solutions. For the Row-Column method the array architecture has been adopted, and for the Vector-Radix method, the pyramid architecture. The texture segmentation algorithm, for which a system-level design is given, demonstrates a further application of the Vector-Radix Fourier transform. A novel concurrent version of the quad-tree based Split and Merge algorithm has been implemented on the pyramid architecture.

The performance of the developed parallel implementations is analysed. Many of the obtained speed-up and efficiency measures show values close to their respective theoretical maxima. Where appropriate comparisons are drawn between different implementations.

The thesis concludes with comments on general issues related to the use of the Transputer system as a development tool for image processing applications; and on the issues related to the engineering of concurrent image processing applications.

Key Words: Parallel Processing, Transputers, Convolution, Image Segmentation, Reconfigurable systems.

To My Wife Akhlis,

and my children; Ahmed, Oscid, Bara and Marowa

Acknowledgements

I wish to express my thanks to:

Dr. Ela Claridge, my associate supervisor, for her support, guidance, and especially her enthusiasm during the writing of this thesis,

Dr. Brain Gay, my supervisor, for all his help and support in publishing and attending conferences,

all my friends, especially Rob Thomson for his advice on aspects of writing style,

Mr. Abdalah AlShawi for his friendship throughout this work,

and finally, I acknowledge the financial support provided by the Iraqi Ministry of Higher Education.

List of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image processing is a multi-disciplinary field involving researchers from diverse domains such as engineering, physics, mathematics, computer science, cognitive science and psychology. The first image processing techniques emerged in the 1920's but until recently the field was mainly oriented towards research and experimentation. This was due to technological limitations in computer storage and speed capabilities required for practical image processing implementations. Recent advances in computer technology, combined with a drop in the prices of processing and memory elements, have made it possible to build image processing systems for real use in industry, surveillance, medicine, meteorology, and many other areas.

In the past the term *image processing* included the acquisition and manipulation of digital images, their compression, enhancement and restoration, as well as their matching, description, interpretation and recognition (Rosenfeld & Kak, 1982). In current terminology this term refers only to processing which is applied to pictorial data and which produces pictorial data as the result; this type of processing can also be called *low-level vision* as it resembles the 'low-level' processes happening on the retina in biological visual systems. Image description (including certain forms of segmentation) and matching techniques are considered to be *intermediate-level vision* processes as they convert pictorial data into an intermediate-level representation which is more suited to symbolic manipulation by higher-level processes. Interpretation and recognition are considered to be *high-level vision* processes. The processing on each level has different computational requirements. Low-level processes deal with large amounts of data; processing involved is usually very simple and homogeneous across the whole image. Intermediate-level processes have still to deal with large pictorial inputs; however processing is more sophisticated and more inhomogeneous, as it may depend on the data being processed. Data volume for high-level processes is relatively small but processes applied may be very sophisticated.

The ultimate goal for many 'real-world' applications would be to combine the processes of all three levels so that images acquired by a camera or other sensing device are processed to yield descriptions or representations useful for a given purpose. Systems with such capabilities are very few at present and are mainly 'state-of-the-art' demonstrators, working within very limited image domains. Low-level and some intermediate-level image processing operations, however, have started to be applied on larger scales to 'real-world' problems.

The practical use of image processing has been hindered by very long processing times, inadequate for real-world applications. One way of achieving an improvement is to introduce parallel processing techniques. The problems encountered in image processing are inherently well suited for parallel techniques and so such techniques can be expected to yield efficient and cost effective results. Earlier attempts to achieve fast response have involved the use of special architectures, capable of real-time execution of tasks such as thresholding or spatial convolution with a small kernel. Such architectures were normally for use on low-level problems and for single techniques. They were not capable of expansion or upgrade as the need arose for intermediate-level image processing applications.

A new tool, the Transputer, appeared in 1985. The Transputer is a general-purpose processor on a chip with memory and in-built parallel communication links.These features make it very suitable for building parallel networks. An additional tool, the software switcher COO4, enables the dynamic re-configuration of the network. Together with the Transputer a new high level language, called Occam, became available. Its novel features were the inherently built in support for concurrency and communications. It was therefore considered appropriate to explore the use of

Transputer-based hardware and Occam for building flexible and extendible practical systems for low- to intermediate-level image processing.

Image segmentation, which is partitioning of an image into its constituent parts, is an important, widely used low- to intermediate-level operation. In its simplest form it is used to divide an image into 'objects' and the 'background'; it also forms an important step in the process of description and recognition of images. As there are many diverse principles of partitioning, their implementation requires the use of many different computational schemes. The segmentation is often termed the 'bottle-neck' of computer vision because of the very intensive computation required to process large amounts of pictorial data. It was therefore seen as a very suitable application area for Transputer research - it offers plenty of challenge and is of practical use.

The challenge was to investigate whether intermediate-level segmentation algorithms with diverse computational schemes could be easily implemented on Transputer systems. This would involve investigating whether Transputer systems would support the different architectures required for these various algorithms. It would also be necessary to explore how well the Occam language is suited for image processing applications. Having implemented a number of Transputer applications it was hoped to compare the efficiency and speed-up improvements achieved for various image sizes. The practical objective of this work was to develop a flexible parallel system for image segmentation at practical speeds, capable of expansion.

1.1 Image Segmentation

There are three broad approaches to image segmentation. In the edge based approach the image is segmented through the detection of *discontinuities* in grey level values; the detected lines and partial edges are then linked to form region boundaries. Within the region based approach the image is segmented into uniform regions thus exploiting *similarities* between grey level values. The texture based approach focuses upon the relationship between texture and tone. There also exist compound schemes which combine more than one approach; some of such schemes are rule-based.

The choice of particular segmentation algorithms for parallel implementation was influenced by following considerations. Firstly, the algorithms chosen should be representative of the three main approaches to segmentation. Secondly, they should represent different computational schemes, so that the flexibility of the parallel system can be investigated and tested. Thirdly, the algorithms should be examples of commonly used image segmentation methods so that comparisons with traditional implementations can easily be made. Thus the segmentation methods chosen were: edge detection by convolution; region segmentation by the Split and Merge algorithm (Pavlidis, 1977); and texture segmentation algorithm by Granlund (1978).

Convolution is the basis of many edge based segmentation methods; it is also used as the main computational mechanism in Granlund's texture segmentation algorithm. It is computationally very intensive and therefore a good candidate for a parallel implementation. Because of its importance amongst image segmentation methods, several convolution schemes were developed and compared.

The Split and Merge method is a frequently used segmentation algorithm, well known for its elegant quadtree decomposition scheme. This scheme has an obvious counterpart in a pyramid architecture, which has recently received a lot of attention as an architecture for computer vision applications (Cantoni & Levialdi, 1986).

1.2 Implementation Issues

The development of parallel processing applications involves the design of both the software algorithms and the overall system set-up which includes elements such as type and number of processors, their interconnections, locations and sizes of memory modules, together with supporting software. On the application level it is important that the architecture and the algorithm are closely and individually matched; on the system level it is desirable to have a single framework for interprocessor activities such as data routing or communication between the processors. Such a framework has been developed and all the application algorithms use the same 'system' procedures. The main area of work, however, was the design and implementation of the chosen segmentation algorithms and analysis of their performance.

There are two methods of implementing the convolution operation; the direct method, carried out in spatial domain; and the indirect method where frequency transforms of an image and a kernel are multiplied together. The image and the kernel are transformed to the frequency domain by applying the 2-dimensional Fast Fourier Transform (2D FFT), and after the multiplication the inverse 2D FFT is applied to produce the resulting convolved image. The choice of the method depends mainly on the size of a convolution kernel.

The direct method was implemented in parallel using an array architecture. This choice of architecture was based on the fact that the direct convolution is a local operation. An image data can be therefore distributed amongst the processors of the array, the kernel sent to each processor and convolution performed in parallel on each subimage. The subimages are overlapped to avoid the intercommunication between the processors and size of the overlapped portion depends on the kernel size.

The indirect method of convolution contains two 2D FFT operations. Two methods considered for parallel implementation of the 2D FFT were the Row-Column method and the Vector-Radix method. The Row-Column method is based on applying one dimensional FFT to each row and then applying one dimensional FFT to each column of the resultant intermediate image. An array architecture was used for its implementation. Each processor in the array contains several rows of an original image. First, a one-dimensional FFT is applied to the rows; this is then followed by a matrix transposition which transposes rows to columns and columns to rows. The second one-dimensional FFT is then applied to the newly formed rows, effectively processing columns of the first one-dimensional transform. On the completion of these operations each processing element contains several rows of a two-dimensional transform of the image. The Vector-Radix method is based on successively applying 2x2 'butterfly' operations to the image. This method was implemented on a pyramid architecture. The three parallel implementations of the convolution operation were compared through analysis of their efficiency and speed-up measures. The comparison includes factors such as image size, kernel size and network size.

The Granlund method is based primarily on convolution of the image with a set of carefully designed kernels which depict textural features of various frequencies and orientations. Each convolution block is implemented on a pyramid architecture. A system-level design of parallel architecture for the full method was developed and performance measures for the whole architecture were estimated from the measures obtained for individual convolution blocks.

The Split and Merge method was implemented on the pyramid architecture since the quadtree structure of the algorithm matches this architecture very well. In addition to the performance analysis for segmentation of a single image, the performance for a sequence of images 'pipelined' through the pyramid was analysed.

Chapter 2

Image Segmentation Strategies

Image segmentation is the operation of partitioning an image into regions or units which are homogeneous with respect to one or more features. It is the crucial stage in the image understanding discipline since the results derived from its implementation are to be analysed by subsequent processing, such as description and recognition. There is still no single theory for image segmentation, despite great research effort which has been put into this task. The existing techniques for image segmentation are usually *ad hoc* and work successfully only for very narrowly specified classes of images. Image segmentation techniques can be categorised into three broad categories: boundary-oriented segmentation, region-oriented segmentation, and texture segmentation. The focus of the boundary-oriented approach is upon discontinuities (nonuniformity) in properties while the region-oriented approach is upon similarities of properties. The focus of texture segmentation is upon the relationship between texture and tone. There is a possibility of using a compound approach which includes the using of more than one category for segmenting broad types of images.

Segmentation processes vary in quite unreliable and unpredictable ways due to a variety of confounding factors such as variations in lighting, perspective distortion, varying point of view, occlusion, highlights, and shadows. Another problem is that objects and their parts may or may not be visually distinguishable from the background. This has led to the difficulty of deriving a precise definition of the uniformity, and the use of the probability statistics to give the approximate uniformity predicates. Errors from the image segmentation process are to be expected because no optimal solution exists in nearly any image segmentation approach.

In this chapter, several techniques for implementing these strategies are described and discussed. Rule-based strategies for image segmentation are also described.

2.1 Boundary-Oriented Segmentation

Image segmentation is defined within this approach as a process of partitioning the image into several closed or nearly closed boundaries which delimit regions. The operation of segmenting images is achieved by detecting the discontinuities and then by linking them. Here some techniques for this type of segmentation are described. Good surveys of the results achieved using this group of methods can be found in Marr (1982), Ballard and Brown (1982) chapter four, and Rosenfeld and Kak (1982) chapter ten.

2.1.1 Edge Detection

Gradient operators are based on the first derivative function to detect the variation. If f(x,y) is the image function, then the definition of the gradient of an image f(x, y) at pixel location (x, y) is a vector G(x,y),

$$G(x,y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix},$$

where Gx represents the variation along the x-axis, while Gy represents the variation along the y-axis. The magnitude of this vector is G(x,y), where

$$G(x,y) = \sqrt{Gx^2 + Gy^2} .$$

The direction of the gradient vector is ß, where

$$ß\,(x,y) = \tan^{-1}\left(\frac{Gy}{Gx}\right) .$$

There are several ways to obtain the partial derivatives at every pixel location. The most common approach is to use first-order differences in a 2 x 2 window as in the Roberts operators (Roberts, 1965); or in a 3 x 3 window like in the Prewitt (1970) and Sobel operators (Duda & Hart, 1973). These operators are shown in figure 2.1.

0	1
-1	0

1	0
0	-1

(a)

-1	0	1
-1	0	1
-1	0	1

1	1	1
0	0	0
-1	-1	-1

(b)

1	2	1
0	0	0
-1	-2	-1

-1	0	1
-2	0	2
-1	0	1

(c)

Figure 2.1 Edge operators. (a) Roberts. (b) Sobel. (c) Prewitt.

Directional operators by Nevatia and Babu (1980), and Binford (1981) introduced implicit averaging which is largely along the edge rather than across it. Nevatia and Babu operators are a set of masks of size 5 x 5 and at 30 angular intervals for linear feature extraction. The choice of the mask size and the intervals was guided by the evaluations of various edge detectors carried out by Fram and Deutsch (1975) and Abdou (1978) and other empirical observations. These masks are convolved with the image and the maximum magnitude of the convolved output and the direction of the corresponding mask are selected to be the edge data.

Isotropic operators by Marr and Hildreth (1980), and Shanmugan *et al.* (1979) offered simplicity and uniformity at the expense of smoothing across edges.

The operators by Marr and Hildreth (1980), and Canny (1983) employed Gaussian smoothing to reduce noise. They used two orthogonal 1-D Gaussian smoothing operations; one along the edge and other across it. Canny described, in an elegant way, the variations based on the Zero-Crossing technique. There are two main features which distinguish this operator from other Zero-Crossing implementations. Firstly, non maximum suppression insures that the detector has only one response to a single edge by defining detection and localization criteria for a class of edges. Secondly, a detection scheme uses several elongated operators along the edges. This is because the step edge detector performance improves considerably as the operator point spread function is extended along the edge. The Canny operator was the first optimal edge detector. Later Spacek (1984) modified the Canny operator by forming a performance measure which combines all three quantitive measures derived by Canny. His optimal filter is different from Canny's. Petrou and Kittler (1988) derived the definitive optimal edge operator by using a cubic spline approximation. The results presented by Petrou and Kittler showed no significant difference between the three operators.

Nalwa and Binford (1986) claimed that for a given support along a locally straight edge Gaussian smoothing is less effective than a simple averaging operation.

Surface fitting operators by Prewitt (1970) and Haralick (1984) employed a means to estimate derivatives. Hueckel (1971) employed a classification technique. They failed to exploit the directionality of the edges. Later Nalwa and Binford (1986) proposed 'step-edgel' detection method based on a surface-fitting approach using directional one-dimensional surfaces. Edgels are edges defined in terms of short linear segments. They claimed that this method is robust with respect to noise.

The test of performance of several edge operators has been made by Suciu and Reeves (1982) on edges of the ramp function. This showed that Prewitt and Sobel operators performed almost equally well but their performance was superior with respect to other local edge operators such as Synder (1980) and other moment operators by Machuca and Gilbert (1981), and Delp and Mitchell (1979).

2.1.2 Edge Linking

The techniques of edge detection are used to detect the discontinuities in the image and produce high pixel values at the boundaries of the regions. However, the natural boundaries are noisy and appear non-continuous due to nonuniform illumination and other effects which produce spurious discontinuities. To get a meaningful set of boundaries, edge detection algorithms are usually followed by other techniques such as linking and further boundary detection procedures. There are two main classes of techniques for that purpose, the local and the global techniques.

Local Techniques

In order to decide whether an edge pixel represents a true or false edge a local information is used. Usually, a small neighbourhood of size 3 x 3 or 5 x 5 is used and within that neighbourhood similar pixels are linked, thus forming a boundary of pixels that share some common predefined properties. Two principle properties are used, the first is the magnitude of the gradient operator used to produce the edge pixel, and the second is the direction of that gradient. The pixels in the predefined neighbourhood are linked if both the magnitude and direction criteria are satisfied.

Global Techniques

As the name suggests the techniques of this kind are based on the global information derived from the data set. Two main edge linking methods in this group are the Hough Transform and Graph-Theoretic method.

Hough Transform

This method, proposed by Hough (1962), transfers the gradient image from the spatial domain into so-called Hough plane, which is a discrete 2-dimensional parameter space usually spanning polar coordinates P(ρ, θ). Each point of this space is then examined for a high value which indicates the presence of a line described by a pair (ρ, θ). By this method a line can be detected and discontinuous edges can be linked. For a recent survey of the work done on the applications and the implementations of the Hough transform see the paper by Illingworth and Kittler (1988).

Graph-Theoretic method

This method was proposed by Martelli (1976) for detecting edges and contours in noisy images. He embedded the properties of an edge in a figure of merit. The detection of an edge in this approach is to minimise this figure of merit. If the edge segments are represented by a graph structure, the search for a shortest path in the graph minimizes the figure of merit and that corresponds to significant edges. This method leads to a substantial improvement in computing time with respect to the Dynamic programming approach (Kaufmann, 1967). However, this method is more complicated and takes more processing time than the local techniques and the Hough Transform method.

2.2 Region-Oriented Segmentation

In this approach the image is segmented into a number of non-overlapping regions which are internally uniform. In contrast to boundary-oriented segmentation, images are segmented on the basis of similarities of local properties. The following sections discuss various techniques within this approach. However, there are other techniques such as measurement space clustering, spatial clustering, thresholding, and others which can be found in many segmentation surveys such as Zucker

(1976), Riseman and Arbib (1977), Fu and Mui (1981), Haralick and Shapiro (1985), and Ballard and Brown (1982) chapter five.

2.2.1 Region Growing

The earliest major approach of region growing was proposed by Muerle and Allen (1968). Its principle is to begin with an initial partitioning, either single pixel regions or several pixels large regions, and then by using a certain feature measure similar regions are merged until no further regions can be merged. Brice and Fennema (1970) applied the state-space approach of artificial intelligence (Nilsson, 1971) to region growing. The process starts by partitioning an image into initial segments, each consisting of a group of pixels of identical grey level. Then the merge algorithm starts to merge the adjacent regions with small differences of grey levels.

Pong *et al.* (1984) proposed a region growing scheme based on the facet model of images (Davis *et al.*, 1975). The process starts with an initial segmentation by grouping pixels using an iteration scheme. Next the merging algorithm follows which has two phases. The properties of each region are updated based upon the properties of its neighbourhood in phase one, while in phase two, adjacent regions with similar property values are merged. The shapes of regions through the process are arbitrary. The thresholded facet iteration method has been used to control two region growing processes which gives the ability to separate inhomogeneous neighbourhood regions.

2.2.2 Splitting

The region splitting algorithm starts with considering the whole image as one region and then applies a certain criteria to split this region into a number of regions. This procedure is applied on each split region until all the regions satisfy the criteria. Ohlander *et al.* (1978) proposed a recursive region splitting method for many different types of images. Histogramming techniques were used to estimate a threshold value for each region under processing. This is done by selecting the best peak in the histogram.

Lee (1986) proposed a hierarchical scope views technique to overcome the two disadvantages of this method, the majority rule problem, and the deficiency if there is no well defined peak in the histogram. A quadtree data structure was used to keep track of the recursive steps and permitted an implementation of the boundary check procedure. This procedure can detect all possible sources of boundary discontinuity, but two problems still remained: the failure to merge regions in the later stages which have been already selected in the early stages of the iteration, and the inability to detect regions containing a large intensity gradient.

2.2.3 Split and Merge

As the segmentation produced by both the region growing (merging) and the region splitting algorithms had serious drawbacks, Horowitz and Pavlidis (1978) proposed a novel solution which overcomes the disadvantages of both algorithms by applying first the split phase and then the merge phase. This technique is called Split and Merge. The full description and implementation of the algorithm will be the subject of chapter eight.

Chen and Pavlidis (1979) applied the Split and Merge algorithm to segment textured images. They used the co-occurrence matrix (see section 2.3.1) as a texture uniformity measure instead of grey level uniformity measures used in the original Split and Merge algorithm. Within this technique the matrices of a region and its four sub-regions are compared. If they are similar, then the region is considered to have a uniform texture, otherwise it is split into four sub-regions. The same criterion is used in the 'merge' phase of the algorithm.

Doherty *et al.* (1986) proposed a further merge procedure following the split and merge method, and called the algorithm Split Merge Merge. The second-pass merge considers all the regions formed by the Split and Merge algorithm, and applies a second-order statistic texture measure (Doherty *et al.*, 1985). This enhanced technique segments various types of images with both low and high textured areas, such a high resolution urban images.

More recently Laprade (1988) proposed a modified technique for the Split and Merge algorithm. This method used a combination of a F-test (Rao, 1973) and a mean predicate to evaluate the uniformity of the regions. The F-test is based on the least square method and compares the residuals obtained by fitting all regions with one plane to the residuals obtained by fitting each region with a plane. If the difference between them is large, the regions are judged to be distinct. Multiple predicates were used because the F-test is not sensitive to the magnitude of the differences between regions, but to their uniformity. The least squares parameters needed for the F-test are performed efficiently by the Split and Merge algorithm because the new parameters for a union of two regions are evaluated by summation of the corresponding parameters of each region, and so they are computed only once.

2.3 Texture Segmentation

Image segmentation according to texture has been acknowledged to be one of the most difficult and time consuming image processing tasks. Image texture has two dimensions (Haralick, 1979), the first dimension is to describe the primitives out of which image texture is composed, and the second dimension is to describe the spatial dependence between the primitives of an image texture. Usually the first dimension is concerned with tonal primitives, while the second dimension is concerned with the spatial organisation of the tonal primitives.

Image texture has another evaluation in terms of so-called the texture features, such as fineness, coarseness, smoothness, and randomness. These need to be translated into some property of tonal primitives and the spatial interaction between the tonal primitives.

The relation between texture and tone is strong and it is similar to the relation between a particle and a wave. The dominant property of a small-area patch of an image which has little variation of tonal primitives is tone, while the dominant property of a small-area patch of an image which has wide variation of tonal primitives is texture. Due to this strong relationship between texture and tone all the existing methods tend to emphasise both, but not equally because of the variety of types of image textures.

There are three principle approaches to defining the textural property of a region and subsequently using this property for segmentation: statistical, structural, and spectral approaches. In the statistical approach, the texture is characterised as smooth, coarse, grainy, and so on (Gonzalez & Wintz, 1988). In the structural approach it is described depending on the regularity of spatial patterns. Spectral techniques are based on the properties of an image frequency transform such as Fourier or Hadamard and the high energy narrow bands in the transformed space are an indication of textural properties. Here the statistical and spectral approaches only are described as the structural approach is used mainly for texture synthesis rather than analysis.

2.3.1 Statistical approaches

These approaches are based on calculating intermediate matrices of texture of the input image depending on the primitive tones and their spatial interrelationships and then calculating the vector of measures. There are various statistical approaches which have been reported in the literature. Good surveys for some of the methods are given in papers by Haralick (1979) and Haralick & Shapiro (1985). The simplest approach for describing texture is to use moments of the grey-level histograms of an image. Others are Spatial Grey Level Dependence Methods (SGLDM) (Read & Jayaramamurthy, 1972), Grey Level Difference Method (GLDM) (Weszka *et al.*, 1976), and Mathematical Morphology (Serra, 1982). Here two of them are described: the Grey-Tone Co-occurrence and the gradient approaches.

Spatial Grey Level Dependence Method

The texture measures computed by histogram based methods with no spatial information are limited because they carry no information regarding the relative position of pixel grey level values with respect to each other. The Grey-Tone Co-occurrence technique does not suffer from this limitation. The structure which contains the required information is called a co-occurrence matrix. The SGLDM is based on this technique.

The co-occurrence matrix is a 2-dimensional array indexed in both directions by a grey level value. The element C(i, j) is a count of the number of pairs of resolution pixels having grey levels g_i and g_j, respectively, and which are separated by a displacement vector $\delta = (\Delta x, \Delta y)$; These values are normalised by dividing C(i, j) by the number of pairs (n) in the image that satisfy δ. Figure 2.2 below shows an example of a sub-image with its Co-occurrence matrix with $\delta = (1,1)$, in this example $1 \leq i, j \leq 3$, g =[0, 1, 2], and n = 9.

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 2 & 0 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 1 & 0 & 0 \end{bmatrix} \qquad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

subimage C(i,j)

Figure 2.2 Subimage and its co-occurrence matrix.

The first step in this technique is to compute a Co-occurrence matrix over each subimage. The next step is to analyse the matrix in order to categorise the texture of the subimage. Haralick (1979) proposed a number of useful descriptors. Some of them are given below; in the formulas C(i,j) indicates a Co-occurrence matrix and Ng is the number of grey levels in the subimage.

a) Maximum probability

$$\max (C(i,j)), \quad i, j = 1, 2, ..., Ng$$

b) Element-difference moment of order k,

$$\sum_{i=1}^{Ng}\sum_{j=1}^{Ng}[(i-j)^k C(i,j)],$$

c) Inverse element-difference moment of order k,

$$\sum_{i=1}^{Ng}\sum_{j=1}^{Ng}[\frac{C(i,j)}{(i-j)^k}], \qquad \text{for } i \neq j$$

d) Entropy

$$\sum_{i=1}^{Ng}\sum_{j=1}^{Ng}[C(i,j) \log C(i,j)],$$

e) Uniformity

$$\sum_{i=1}^{Ng}\sum_{j=1}^{Ng} C^2(i,j),$$

He *et al.* (1987) proposed a new approach to texture feature extraction from a Co-occurrence matrix. They proved that the computational time required by their method is much less than that required by the traditional Co-occurrence matrix method. The proposed features are calculated and compared with those suggested by Conners and Harlow (1980). The three proposed features are,

a) Diagonal Moment

$$D(\mathbf{v}) = \frac{\sum_{i=1}^{p} i\, S(i,i,\mathbf{v})}{\sum_{i=1}^{p} S(i,i,\mathbf{v})}$$

b) High Level Moment

$$H(\mathbf{v}) = \frac{\sum_{j=D(\mathbf{v})}^{p} [(j - D(\mathbf{v}))\, S(D(\mathbf{v}),j,\mathbf{v})]}{\sum_{j=D(\mathbf{v})}^{p} S(D(\mathbf{v}),j,\mathbf{v})}$$

c) Low Level Moment

$$H(\mathbf{v}) = \frac{\sum_{j=1}^{D(\mathbf{v})} [(D(\mathbf{v}) - j)\, S(D(\mathbf{v}), j, \mathbf{v})]}{\sum_{j=1}^{D(\mathbf{v})} S(j, D(\mathbf{v}), \mathbf{v})}$$

where P is the number of grey levels in the original image, and $S(i,j,\mathbf{v})$ is the estimated probability of going from grey level i to grey level j given the displacement vector $\mathbf{v} = (Dx, Dy)$, assuming that

$$S(i, j, \mathbf{v}) = S(j, i, \mathbf{v}) \quad \text{(i.e. co-occurrence matrix is symmetric)}$$

These features are used as parameters in Bayes' discriminate functions (Duda and Hart, 1973). On 25 randomly selected samples over each Brodatz's (1966) texture samples the correct classification result was 93%. The six features of

Conners *et al.* (1984) lead to a higher correct classification rate (97%), but took much longer time than that of He *et al.*.

Many researchers have used this technique for texture segmentation of various types of images, examples include Haralick *et al.* (1973) and Chien and Fu (1974).

Texture gradient Method

This method is based on the measurement of the grey level run length. A grey level run is defined as a sequence of linearly adjacent pixels having the same grey level value. The length of the run is simply the number of pixels in the run. The grey level run length is a texture primitive, so for coarse textures there are many pixels in a constant grey tone run, while for fine textures there are few pixels in a constant grey tone run. Galloway (1975) described the texture in four grey run length matrices. Each matrix evaluated one principal direction. The grey level run length method is based on computing the number of grey level runs. Each entry (i,j) in the matrices denotes the number of the runs with grey level i and length j along the associated direction. This method is suitable for linearly structured texture.

Later Wermser (1984) described an unsupervised segmentation technique using a texture gradient. In this method a feature vector was constructed for each point in the image and consisted of four feature measures: short runs emphasis, long runs emphasis, run length distribution, and run percentage. Each feature is computed for four directions (0, 45, 90, 135 degrees) so a vector of length 16 is computed. These are evaluated by 16 separate matrices.

Powley *et al.* (1986) applied this technique for the localization of microstructural regions. They concluded that texture gradient method proved to be difficult to use due to the large number of parameters required and the necessity to tune them to the image data. These parameters are the sizes of the run length sampling window and the size of the gradient subwindow.

2.3.2 Spectral approaches

Power Spectrum Method (PSM)

The use of a power spectrum for texture segmentation is widely spread due to the strong relation between the distribution of the power spectrum and the arrangement of patterns in the textured images. For example, for a fine-grained and closely spaced regions the high values in the power spectrum will be spread out away from the origin, while for coarsely grained regions the high values in the spectrum will be concentrated close to the origin. The direction of the texture can be detected by the power spectrum as well: for example, horizontal streaks in the region will give rise to a vertical streaks in the spectrum (Rosenfeld & Weszka, 1976).

The first step is the computation of the sample power spectrum Q(u, v):

$$Q(u, v) = |F(u, v)|^2,$$

where F(u, v) is the Fourier transform of the image. The best representation of power spectrum is by using polar coordinates,

$$\phi_p(\rho, \theta) = Q(u, v),$$

where $\rho = \sqrt{u^2 + v^2}, \theta = \tan^{-1}(v/u)$.

The integration of ϕ with respect to θ and ρ give the functions $\phi_1(\rho)$ and $\phi_2(\theta)$ respectively: $\phi_1(\rho)$ represents the spread of values of $\phi(\rho, \theta)$, while $\phi_2(\theta)$ represents the directional biases of $\phi(\rho, \theta)$. There are three commonly used features with power spectrum (Weszka *et al.*, 1976):

1) Annular-ring sampling geometry: These can be expressed in polar coordinates as

$$a_j = \int_{\rho}^{\rho_j+\Delta\rho} \int_0^{\pi} \phi_p(\rho, \theta)\, \rho\, d\theta\, d\rho, \quad j = 1, 2, \ldots, m_a$$

where $\phi_p(\rho, \theta)$ is the sample power spectrum with polar coordinates, and m_a is the number of annular rings.

2) Wedge sampling geometry: These measurements can be computed by the following expression:

$$a_j = \int_{\rho_{min}}^{\rho_{max}} \int_{\theta}^{\theta_j+\Delta\theta_j} \phi_\rho(\rho, \theta)\, \rho \, d\theta \, d\rho, \quad j = 1, 2, \ldots, m_w$$

where m_w is the number of wedges.

3) Parallel-slit sampling geometry. Assuming that the power spectrum is first rotated by an angle θ, the slit sampling geometry can be evaluated by

$$s_j(\theta) = \int_{-v_{max}}^{v_{max}} \int_{u_j}^{u_j+\Delta u} \phi(u, v)\, du\, dv, \qquad j = 1, 2, \ldots, m_s$$

where m_s is the number of slits.

Hadamard Transform Method

The Hadamard Transform (Pratt *et al.*, 1969) decomposes a function by a set of orthogonal waveforms. The 2D Hadamard Transform H(u, v) can be expressed as

$$H(u, v) = \sum_{x=0}^{N-1} \sum_{Y=0}^{N-1} f(x, y)\, (-1)^{p(x, y, u, v)},$$

where

$$p(x, y, u, v) = \sum_{i=0}^{L-1} [bi(u)\, bi(x) + bi(v)\, bi(y)],$$

where bi(r) is the i-th bit of the binary representation of the operand r and NxN is the sub image size where $N = 2^L$.

A texture feature proposed by Powley *et al.* (1986) is extracted from the transform of a 16 x 16 local area around each point in the image. The value of the texture feature at each point is computed as the sum of the differences between the row and column totals of the local area transform. The sum of each row and column are R and C respectively and they are evaluated as follows,

$$R(u) = \sum_{v=0}^{15} H_{ij}(u, v),$$

$$C(v) = \sum_{u=0}^{15} H_{ij}(u, v),$$

where H_{ij} is the Hadamard Transform of the 16x16 local area centred on the image pixel at location (i, j). The texture measure Tm for the pixel location (i,j) can be evaluated by the following equation,

$$Tm = \sum_{r=0}^{15} [R(r) - C(r)].$$

This texture measure is calculated for each pixel in the image. The difference between the symmetry of the transform taken across different regions can be distinguished because the texture measure is symmetric for the boundary while it is asymmetric for the area internal to the boundary.

The Hadamard method provided best results on microstructural regions in the applications of Powley *et al.* (1986). It has the advantage of simple implementation and so runs faster than the gradient method. The results of the Hadamard method can be used in post-processing techniques such as thresholding and line thinning for best segmentation.

Granlund method

This method has been developed by Granlund and colleagues at Linkoeping University, Sweden, its details can be found in a number of papers (Granlund, 1978; 1980a). It relies on combining a sequence of image transforms into a hierarchical structure in which each transform contains information related to a certain frequency band. The transforms are derived by convolving an image on each level of the hierarchical structure with a convolution mask (operator) whose size becomes increasingly larger on ascending levels of the hierarchy. The method is based on the assumption that within an image window of a certain size there is a dominant frequency band in a certain direction. Thus by applying an operator of an appropriate size within a window a vector can be calculated with a magnitude which represents the dominant frequency (variation) within that window and a phase which represents the direction of that variation.

The first order transform is derived by replacing each pixel in the original image by the vector with the largest magnitude, selected among the vectors calculated for a number of directions for the window centred at that pixel. The first order transform is then added to the original image and a subsequent transform, with the larger operator size, is derived to detect lower frequencies than those on the previous level. The obtained image is a second order transform and further transforms are evaluated in the similar manner. The authors refer to the operator as 'the general picture processing operator' and see its application, among others, in separating regions of different texture. The full description of this method and its implementation is the subject of chapter seven.

Gabor modulation/demodulation

Clark *et al.* (1987) used a two-dimensional Gabor filter (Gabor, 1946) to segment images into regions of specific frequency or orientation characteristics. The filter is tuned to the characteristics of the regions. The image is transformed into a narrow band signal by these filters. The envelope of the transformed image coincides with the regions whose characteristics the filter is tuned to.

The properties of the Gabor filters make them useful for finding texture boundaries. The useful properties include optimal localization in space and frequency, the ability to tune the bandwidth and the band location and the ability to use the filters as a means for modulating textures where the region containing the texture is regarded as an envelope which can be recovered.

Clark's paper suggests that phase modulation (PM) could be performed on the narrowband complex images obtained by Gabor filtering. The phase information can then be used alongside with the gradient information as the abrupt changes in phase indicate perceptual texture boundaries, and smoothness in phase indicates smoothness in surface characteristics.

2.4 Rule-Based Segmentation

Rule-based systems have been used for image segmentation and recognition. Such a system contains two main subsystems, the knowledge and control systems. The first proposal to apply this to pattern recognition came from Sloan (1977); other system are reported by Brooks *et al.* (1978) and Levine & Shaheen (1981). All these approaches have concentrated on high level processing. The first Rule-based system for low level processing was proposed by Nazif and Levine (1984), followed by a later system by Stansfield (1986). Here these two systems are briefly described.

2.4.1 Nazif and Levine System

This system has been proposed by Nazif & Levine (1984) for image segmentation. The system is composed of three parts as shown in the block diagram of Figure 2.3 overleaf.

Short Term Memory (STM) is used to store the input image, the segmentation data, and the output of segmentation processing. This data can be read and modified. Long Term Memory (LTM) stores the Rule-Based Model. This data can be read only since the Rule-Based Model is fixed.

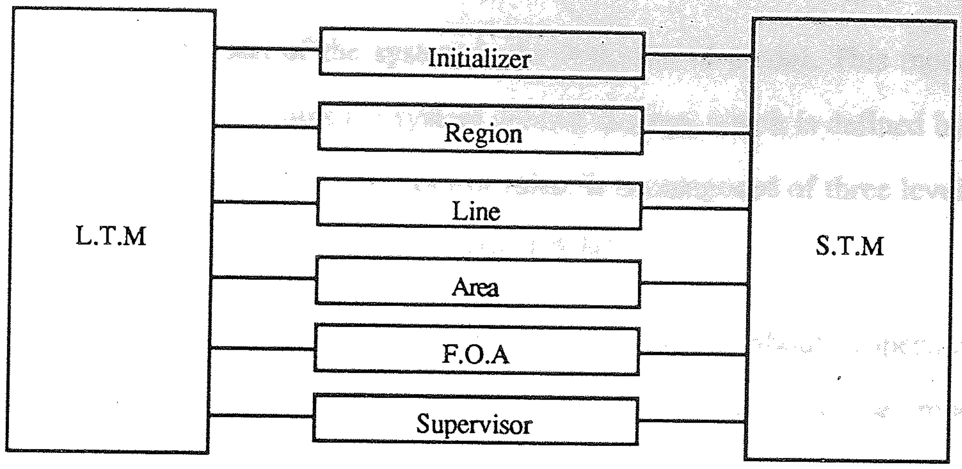


Figure 2.3 Nazif & Levine system.

During segmentation the data in the STM is modified according to the rules stored in the LTM. The modification is performed by an appropriate process chosen according to the matched rule. The following processes are available:

Initialiser :- It generates the initial region and line maps using traditional methods of region and edge based segmentation. By using those maps, it then produces the initial focus of attention areas, and finally computes the performance measurements for each area (Levine & Nazif, 1985b).

Region, line and area processes :- The region process matches the region analysis rules in the LTM to the data in the STM; if the rule fires, the action specified by that rule is executed on the current region under consideration. Similarly the line and area processes execute action on lines and areas respectively.

Focus of Attention process:- It selects the particular data to match the control knowledge rules by using focus of attention rules which are stored in the LTM.

Supervisor process:- It acts as a monitor for system control purposes. It matches its own metarules to the data and consequently determines the order of activity of the other processes.

The most important part of the system is the Rule-Based model. This model is stored in the LTM. It contains the system control strategy which is defined by the strategy rules and executed by the control rules. It is composed of three levels of production rules as follows (Levine & Nazif, 1985a) :

1. The Knowledge Rules:- They encode the information about properties of regions, lines and areas in the form of a set of situation-action pairs. They modify the data depending on the criteria embodied within them. Their actions are :-

a) Merging and splitting regions.

b) Adding, deleting, joining and connecting lines.

c) Creating and updating focus of attention areas.

2. The control rules :- These are two sets of rules as follows:

a) Focus of attention rules:- They specify the next data item to be tested. This strategy is a data-driven process since the condition of these rules depend on the STM data. These rules modify the STM data.

b) Metarules:- They examine the data in the STM, and their actions specify the next process to be activated, and hence the next knowledge rule to be matched. They are responsible for evaluating the stopping criteria of the system and halting the processing procedure. They do not modify the STM data.

3. Strategy rules:- Their function is to select the set of rules that execute the most appropriate control strategy depending upon the data. This selection is based on a set of performance measurements computed for each focus of attention area in the image. This strategy is a data driven one.

This approach combines different methods to create a knowledge system and control strategy to produce a complete production system. This set-up gives two important advantages (Nazif & Levine, 1984): efficient processing by ordering the areas in the image to be processed and minimising the segmentation errors by choosing the appropriate rule for each step of processing.

An optimum set of rules was proposed and tested on a number of images and the performance of the system was compared with that of the histogram-based segmentation (Ohlander, 1975) and Split and Merge (S & M) methods. The results showed that histogram-based segmentation produced highly contrasting regions that exhibited low uniformity values, the S & M produced highly uniform regions with little contrast across the boundaries, while the Rule-Based system produced regions with high uniformity and contrasting. The main drawback of this method is the complexity of the system and that it takes a long time to segment an image compared with other two methods (Levine & Nazif, 1985b).

2.4.2 ANGY

ANGY is a rule based expert system for automatic segmentation of coronary vessels from subtracted digital angiogram of the chest. ANGY has been designed and implemented by Stansfield (1986). This system is modularised into three stages. The first stage is the preprocessing stage which creates a segmented representation of the input image. The second stage is the low-level image processing stage. This stage embodies a domain-independent knowledge of segmentation, grouping, and shape analysis. It attempts to refine the segmentation begun by the preprocessing stage. The third stage is the high-level processing stage. It embodies a domain-dependent knowledge of cardiac anatomy and physiology. The overall system is similar to that of Nazif and Levine (1984) but uses different measures and rules.

2.5 Summary

Image segmentation is a ubiquitous operation which is one of the bottle-necks of image and computer vision processing. This is due to a high volume of data involved in the processing. Three broad categories of image segmentation methods have been identified. The boundary oriented methods and the region oriented methods aim to partition an image according to its tonal composition; texture segmentation methods aim to partition an image into regions of similar texture. There exist also schemes which combine these basic three approaches in either *ad hoc* or rule based framework.

A survey of image segmentation methods, briefly summarised in this chapter, was conducted to help to identify general classes of computational schemes widely used for image segmentation. Two such schemes emerged: convolution and regular decomposition.

Convolution is used in many segmentation methods, especially those which are edge based. It is also used in other (non-segmentation) image processing operations such as smoothing and enhancement. Convolution is also at the heart of a successful texture segmentation scheme by Granlund.

Regular decomposition is an example of a divide-and-conquer strategy and is the basis of the Split and Merge segmentation algorithm and many multi-resolution schemes. The Split and Merge algorithm is widely used for region based segmentation since it provides a general mechanism in which segmentation (uniformity) criteria can be changed at will. It can be used for both tonal and textural segmentation.

Both of these two mechanisms had potential for parallel implementation and at the time of starting this work there had not been many such implementations reported in literature.

Chapter 3

Parallel Architectures and Performance Evaluation

Parallel processing has the capability of providing fast and flexible solutions to computationaly intensive tasks. The image processing domain presents many such tasks. The segmentation algorithms reviewed in the previous chapter present a suitable challenge as they have to deal with a large amount of pixel data and represent an important class of image processing algorithms.

Parallel processing applications require the design of both, the parallel algorithms to suit the architecture of the computer, and the parallel architecture, to best suit the algorithm. For many years the architecture aspect of design involved either simulation by software on a single processor machine, simulation by a network, or building a special hardware.

This work began when it had just become possible to build a parallel architecture from off-the-shelf components - Transputers. A high level parallel language Occam and the associated development tools had also became available. The work carried out during this project must be put into perspective by looking at the history of development of generations of image processing systems and classes of parallel processing systems.

This chapter begins with an overview of image processing systems grouped (after Cantoni & Levialdi, 1988) into four generations; the basis of grouping is mainly chronological but also entails other factors such as methods of image acquisition, distribution and processing. Further in this chapter a well known classification of parallel architectures, on the basis of multiplicity of instruction and data, is given and for each class examples of existing parallel systems are discussed. Methods of evaluation of parallel systems are outlined in the last part of this chapter. Such methods are essential both for assessing relative merits of individual parallel implementations and also for drawing comparisons between implementations of the same algorithm on different architectures.

3.1 Generations of Image Computers

The advances in parallel processing techniques are dependent on the advances in the technology. Parallel processing techniques include architecture design and programming skills. Architecture design includes processors, memory, interconnection network, and control.

There are three vital factors for a better parallel processing system: the speed of processor, the speed of the memory access, and the speed of the communication between the processors in the network. All these factors are strongly related to the technology. The more advanced technology, the more speed can be achieved.

The question arises: why are parallel processing techniques waiting for more new advances in technology to jump to the next stage?

There is no clear answer to this question. However, there are several partial answers ;

1) The high speed of the evolution of the technology gives little chance to the researchers to exploit and fully utilise the existing technology. Here the demand on the applications side is such that it is better to use the new technology to get more performance and hence more profit.

2) If the existing technology is not fully utilised then there is no point of trying to utilise and exploit the future technology, since there is a possibility of getting better performance by investigating the existing technology.

3) The difficulties of getting more grants for future technology. This is clear by comparing the large gap between the research on the existing technology and the research using modelling techniques which are mainly concerned with future technology.

Image processing and computer vision include some of the most computationally intensive tasks. The need for high performance computers for these fields is clear. That is why the advances in computer vision are strongly related to and coincide with advances in parallel processing techniques. Since parallel processing techniques are strongly related to the technology the advances in computer vision are strongly dependant on the technology and the generations of the image computers are the same as those of the technology. Figure 3.1 shows graphically the relationships between the three advances in computer vision, parallel processing, and technology.

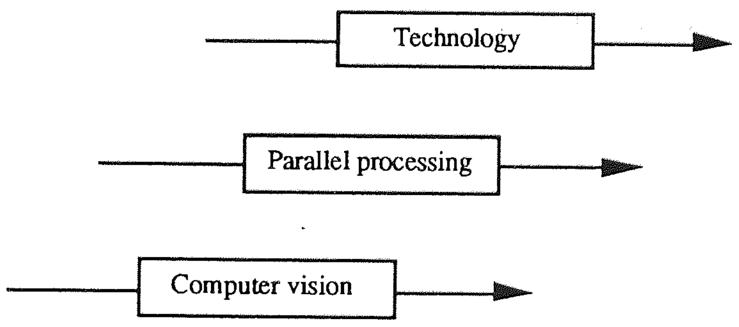


Figure 3.1 Advances in the computer vision three related fields.

3.1.1 First Generation 1955-1965

In this period the sequential raster scanning of the image is the first data acquisition method. This is achieved by scanning the image row by row and storing the pixel information in a buffer memory which is capable of storing only portion of the image. The operation is very slow since only a portion of the image can be processed, so there is a need to have multiple access to the image memory and overhead of row overlap and large number of image segments.

In this period, Fortran programming language was the only high level tool available. For performance improvement, a special low level machine language could be used for each hardware block.

The typical image processing achievements of this period were image enhancement and restoration. Image enhancement is the operation of increasing the signal to noise ratio, for example by low pass filtering to suppress the high frequency noise or band pass filtering to enhance edges.

3.1.2 Second Generation 1965-1975

In this period intensive research was done towards implementing local operations such as histogramming, template matching, and connected component extraction.

To achieve such operations, all neighbouring pixel values of an elementary 3 by 3 subarray were simultaneously provided to a special function unit. This needs a full scanning of the image. Examples of computers of this period are GIOPR (Preston, 1971) and PICAP I (Kruse, 1973).

3.1.3 Third Generation 1975-1985

In this period multiple computational units became available. These are configured in one of three classes:

Processor array computer

These computers used a 1-bit processor as a basic block to the array. The processing elements contain an arithmetic logic unit (ALU) and a special logic to implement the bidirectional interconnections with the nearest neighbours. Examples of such computers are MPP (Batcher, 1980) and CLIP4 (Duff, 1978).

Pipelined computers

These machines contain up to 100 processor elements connected in a pipeline. These systems require an accurate balance of tasks between the stages. An example of such a machine is the cytocomputer (Lougheed *et al.*, 1980).

Multiresources computers

These computers use a multiresources management strategy for deriving and coordinating the activity of a special function unit by a host computer. The special function unit is a hardware device corresponding to the standard routines for histogramming, template matching, convolution etc. Examples of such machines are PICAP II (Kruse *et al.*, 1980) and TOSPICS (Mori *et al.*, 1978).

3.1.4 Fourth Generation 1985-

In this period an emphasis was given to the types of connection between the processors so as to increase the flexibility of the architectural design; and the possibility of reconfigurable systems to provide static and dynamic hardware matching to the task under processing (Cantoni & Levialdi, 1983). This approach uses an interprocessor router to map image data structures and data flow during computation. An interesting reconfigurable system is SIMD/MIMD (Siegel *et al.*, 1981). These architectures are so-called multicomputers due to autonomous capabilities of the new single processing units (Uhr, 1987).

In these architectures, programming becomes a complex task since concurrency, multiple resources management, temporal loading variations and dynamic dispatching must all be handled:

-Different programming paradigms are necessary, due to the use of a variety of processing modes, such as procedural, object oriented, and declarative;

-The possibility of a hierarchical organisation (Tanimoto & Klinger, 1980) of computation levels may benefit from the planning strategy which should be dynamically generated;

-The man-machine interactivity and system programming must be comfortably supported by visual/iconic interfaces which also allow the direct execution of the target task by iconic programming (Chang, 1987).

The complexity of these systems makes their tuning to the various computer vision applications difficult to achieve. Knowledge of these system capabilities as well as of the task domain is required to achieve high performance, fully exploiting the available resources.

3.2 Taxonomy of parallel architectures

It is clear from the previous section that there are various types of architectures proposed for image processing and computer vision applications in general. This is due to the complexity of these applications and the limitations imposed by the hardware technology. This means that there is no single type architecture which is suitable for all types of applications, i.e., the parallel architecture is applications dependant. The direction of research in parallel processing is towards designing a parallel architecture for the maximum possible number of applications.

Flynn (1966 & 1972) has proposed the best known taxonomy for parallel architecture computers. It is based on the multiplicity of instructions and data. According to this basis, parallel architecture can be classified into four groups:

SISD- Single Instruction stream Single Data stream computers which represent the Von Neumann conventional architecture in which there is a single program controller in which each single instruction is executed on a single datum.

SIMD- Single Instruction and Multiple Data computers in which there is a single controller which stores the program. This controller broadcasts a sequence of instructions to a number of processing elements (PEs), each of which executes these instructions on its own memory.

MISD- Multiple Instruction stream Single Data stream computers, usually called pipeline computers. In this type several instructions are executed on a single datum.

MIMD- Multiple Instruction stream Multiple Data stream computers. In this type several instructions are executed on several data items.

This type of taxonomy is not sufficient for the recent advances in technology. Thus the need for more than the multiplicity of instruction and data to be included such as processor autonomy (Fountain, 1987).

In this section, the parallel architectures for image processing and computer vision are classified according to their multiplicity of instructions and data. The processor autonomy is used to further classify SIMD computers.

3.2.1 SIMD

As stated, these are Single Instruction stream-Multiple Data stream machines which consist of a two-dimensional array of relatively simple processing elements (PEs). There is a local connectivity between these processors and local memory associated with each of them. A central controller (host) broadcasts a sequence of instructions to the PEs, each of them executing the same instruction simultaneously but operating on its own local data. The other important part of SIMD is the interconnection network which allows data to be transferred among the PEs. SIMD architecture is obviously well suited to exploiting the parallelism inherent in certain tasks performed on vectors and arrays (images). This type of architecture is quite useful for the following two reasons. Firstly, the single instruction stream aspect makes many of the techniques and practices developed for programming conventional serial computers directly applicable; secondly the parallelism is in the execution of a single task, not in multi-tasking.

Great attention should be paid to the network interconnection because the PE-to-PE transfer of information must be efficient, otherwise the parallel algorithms will be slowed down. There are many SIMD systems, for example DAP (Hunt, 1981), CLIP 4 (Fountain, 1981), and others (Kittler & Duff, 1985).

This leads to further classification of SIMD machines according to their autonomy, as proposed by Fountain (1987), and then by Maresca *et al.* (1988). They identified three types of processor autonomy: operation autonomy, address autonomy, and connection autonomy.

Processor autonomy can be defined as the degree of freedom given to the processor for the three activities of operation, addressing, and interconnection, to be different from other processors activities. Three different autonomies can then be defined (Fountain, 1987). In operation autonomy different operations may be executed by the processors in the parallel computer. This removes the restriction imposed by SIMD computers that a single instruction should be executed by all processors. The addressing autonomy overcomes the SIMD limitation that all the PEs fetch operands from the same address. It gives the freedom to the PEs to access them from their memory. The interconnection autonomy overcomes the limitations of several parallel computer architectures of static interconnection to a dynamic one in which the topology of the network may be changed during program execution.

3.2.2 Non autonomous SIMD computers

There are two types of this group of computers, classed according to the length of the operand, the bit-serial and the integer computers.

Bit-serial computers

These computers contain a single control unit (CU) and an array of identical PE's. The CU distributes the whole image or consecutive blocks of the image through the PE's, to be processed in parallel. Each PE is connected to PE's adjacent to it and the communication between them is only one bit wide. The main strategy of operation is that each PE fetches data from its own memory or any of its directly connected neighbouring PE's memories, and executes a sequence of instructions sent by the CU.

The bit-serial architecture is very efficient with respect to memory and processing resource utilisation because of the flexible data formats allowed by this type of architecture (Reeves, 1984).

Most of the computers built using this type of architecture are array topology, and they have over 16000 PE's. Examples of these are the 64-by-64 DAP (Reddaway, 1978 & 1980), the 96-by-96 CLIP4 (Fountain, 1981), and the 128-by-128 MPP (Batcher, 1980) whose novel feature in the PE was the dynamically reconfigurable, variable length shift register with a maximum length of 30 bits. This organisation allows very fast multiplication because the shift register is used for circulating the partial product.

Integer computers

These computers are similar in operation to the bit-serial type, with only one difference: the PE is of several bytes size instead of 1 bit. An example of this type is NON VON (Shaw, 1982).

3.2.3 Autonomous SIMD Computers

Maresca *et al.* (1988) applied a recently proposed autonomy to further classify fine-grain SIMD massively parallel processing. These are operation autonomy, addressing autonomy, and connection autonomy. They concluded that connection autonomy is the major factor in the development of massively parallel computers for vision.

Operation Autonomy

Operation autonomy provides a massively parallel architecture with the ability to execute more than one operation among all PE's. Examples of such machines are the CLIP 7 system (Fountain, 1986 & 1987), and MSIMD computers where several program controllers exist and each of them is associated with a cluster of PE's, for example in a pyramid (Cantoni & Levialdi, 1987).

Addressing Autonomy

This type of autonomy allows each processor to generate an address locally or modify the broadcast address independently. Examples are ILLIAC IV (Bouknight *et al.*, 1972), CLIP 7 (Fountain, 1987), GF11 (Beetem *et al.*, 1985), and WARP (Annaratone *et al.*, 1986).

Connection Autonomy

Connection autonomy is the operation of efficient mapping of the task graph derived from a vision algorithm and underlying hardware network topology, via dynamic network configuration. Its goals are efficient graph embedding, high reliability, and high availability. There are two types of connection autonomy: packet switched and circuit switched.

Packet switched connection autonomy.This allows each PE to send messages to any other PE in the system by simply sending the destination address in the head of the message and relying on a router at each PE to handle the message. The Connection Machine (Hillis, 1985) is probably the best known example of a massively parallel computer featuring packet switched connection autonomy.

Circuit-Switched connection autonomy. This is suitable for a high-degree network (e.g. hypercube). It dynamically establishes interconnection paths between non-neighbouring processors by crossing intermediate processors. It has the disadvantage of high wiring cost. In addition, its potentially greater flexibility may not be required for much vision computation, which is characterised by regular, fixed task graphs. The Polymorphic-Torus machine (Li & Maresca, 1987) is an example of this type of system.

3.2.4 MIMD

These are Multiple Instruction stream-Multiple Data stream machines. This type was proposed by Flynn (1966). It consists of P processors and M memories, where M≥P. Each processor can follow an independent instruction stream. As with SIMD machines there is a multiple data stream and an interconnection network by which each processor can communicate with the other processors. There may be a control unit responsible for the overall coordination of the activities of the processors.

The main feature of a MIMD machine is the possible increase in the system performance in order to satisfy the applications requirements. For example in image segmentation there are some algorithms which perform the same operations for all the pixels (eg.thresholding) while others contain different operations for each group of pixels; hence the necessity of MIMD. It is clear that MIMD machines are more complex but more flexible for different applications with respect to SIMD machines.

The ideal characteristics for a general purpose MIMD system are as follows (Reeves, 1986):

Extensibility: It should be possible to add processors as demands on the system increase, just as one now adds additional memory to processors.

Fault tolerance: It should be possible to reconfigure the system if any component becomes faulty so that the computation can continue with a minimum of interruption.

Programmability: The user can write a program in a high level language without knowing how many processors the system has.

The main problems in the design of the MIMD system are the subtask allocation strategy and interconnection network design.

There are three types of MIMD architectures Loosely Coupled (low interaction), Moderately Coupled (medium interaction) and Tightly Coupled (high interaction).

Image processing applications are implemented on tightly coupled architectures because they need high speed interconnections between the processor elements. There are many MIMD machines, for example PICAP II (Kruse *et al.*, 1982), EMMA (Manara & Stringa, 1981), Multicluster (Reeves, 1985), and others (Kittler & Duff, 1985).

Image segmentation algorithms need both SIMD and MIMD architectures. Hence a machine which permitted both modes of operation may form an optimal solution to the image segmentation problem. PASM (Siegel *et al.*, 1981) and PUMP (Briggs *et al.*, 1982) represent the most common types of such machines.

Notably, two parallel segmentation algorithms have been successfully developed on PASM. These are edge-guided thresholding (EGT) and contour tracing (Siegel *et al.*, 1981). The designers of PASM concluded that analysis of the algorithms has motivated the inclusion of several important architectural features. These features were used to discuss possible configurations of a custom-designed VLSI processor chip for PASM. Also the use of algorithm characteristics to drive the design of PASM leads to a machine with features that provide the necessary flexibility for executing image and speech processing algorithms.

3.2.5 Pipeline architecture

This type of parallelism can be very economical (Hwang & Briggs, 1985). The basic idea is to subdivide the process into a sequence of subprocesses, each of which can be executed by a specialised hardware stage that operates concurrently with other stages in the pipeline. Many processes are streamed into the pipe and get executed in an overlapped way at the subprocess level. Three main advantages of this architecture are that: it is not necessary to reformat the input data; interconnection between processing stages are very simple; a high speed control unit is not required since an instruction resides in a stage for many clock cycles.

The pipeline computers are especially useful in morphological image processing where many local operations are performed on a given image (Rosenfeld, 1988). The most common pipeline computer is the Cytocomputer first proposed by Sternberg (1979). Also there is special hardware system designed by Douglass (1982).

Cytocomputer

The cytocomputer is a serial pipeline of programmable processing stages. Each stage performs a single transformation of the processing sequence on an entire image. Images are fed into a cytocomputer in a line scanned order and progress through the pipeline of processing stages at a real time rate (speed of data collection). This is achieved presumably after an initial delay to fill the pipeline.

Cytocomputer operations are implemented in highly efficient cellular computer architectures (Sternberg, 1983). Image processing algorithms are constructed using image processing algebra. Image algebra is the formulation of cellular computer image processing algorithms into algebraic expressions whose variables are images and whose operations logically or geometrically combine images.

The pipeline configuration is used instead of the array configuration because in case of using cellular array architecture for image processing as many cells are required in the array as there are pixels in the image. As a result this will be very costly, complex to implement and less flexible, especially for large images, with respect to the pipeline configuration. The block diagram of the cytocomputer is shown in figure 3.2 overleaf.

There are two types of pipeline stages in the cytocomputer, the 2-dimensional pipeline stages and the 3-dimensional pipline stages. In the first, image algebra operations are applied to planar binary images (a binary image has only two pixel intensity values for black and white). In the second, transformations of the same set of operations are performed on grey-scale images.

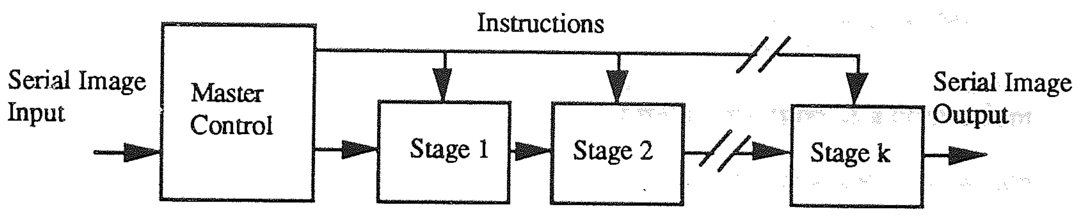


Figure 3.2 Block diagram of the cytomputer.

The main drawback of the cytocomputer is that for applications where the number of operations needed is more than the number of stages, the length of the required pipeline is accommodated by reprogramming the available stages while the partially processed image data is temporarily stored on disk. This will slow the speed of operations.

Special hardware system

A system suggested by Douglass (1982) is an example of a special hardware system built specifically for image segmentation. He chose Yakimovsky's algorithm (1976) as a region growing method for segmentation. He showed that this process can be implemented in a five stage, eight process pipeline using cheap, off the shelf microprocessors to attain several orders of magnitude increase in segmentation speed at a modest cost. He concluded that the pipeline architecture offers a significant increase in speed while avoiding the complex interconnection schemes required for a network approach to segmentation and the memory contention problems inherent in the multiprocessor approach. He suggested that the processing could be speeded up by implementing the pipeline using VLSI technology and very high speed memory.

3.3 Multiresolution Systems

A multiresolution approach has been recognised for a long time as a convenient structure for both image representation and implementation of algorithms which use 'divide-and-conquer' strategies (Stout, 1986). There are three types of architectures for its implementation.

3.3.1 Multiresolution architecture

Burt *et al.* (1986) have proposed a pipeline approach to achieve a multiresolution pyramid. It is based on using five special computational units such as filter, a decimator, an expander, ALU, and memory. Figure 3.3 shows a Gaussian pyramid construction.

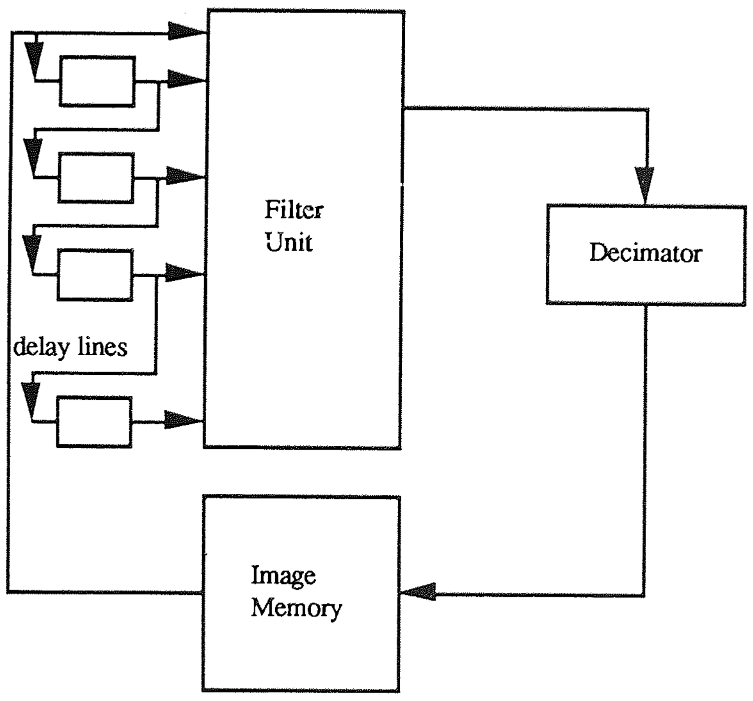


Figure 3.3 Gaussian Pyramid block diagram.

The original image is first fed from image memory to the filter unit row by row. The window size used in the filter unit is 5 by 5, so five rows at a time are available to it. This is achieved by the five FIFO (first-in first-out) delay lines. The output of the filter is passed to the decimator which discards every other row and column of data. This result of the next level of pyramid which is less in data by a factor of 2x2, i.e. coarser.

This machine has been built to achieve real-time performance for applications such as, for example, the surveillance of buildings and their environment in an outdoor scene using coarse–fine search (foveation).

This architecture is cost effective but it is too slow in comparison to ordinary pyramid structure. The main two advantages of this design over the ordinary design; it provides greater processing power per unit cost and it is more efficient and flexible for window processing.

This type of architecture can be classified as special purpose hardware. However, the foveation strategy is of great interest within "active vision" (Brown, 1989), which is an important new field of research within image processing.

3.3.2 Pyramidal Architecture

A good solution to the multiresolution (hierarchical) problems is the pyramid. This is because of the strong structural relationship between the two. Moreover, it allows fast communication between any two processors. It is suitable for implementing both bottom-up image analysis and top-down image analysis.

A pyramid with (N+1) levels (where $N \geq 0$) has 4^N elements on its base level which is called the size of the pyramid, 4^{N-1} on the consecutive level and 1 element on the top (root) level. Each element is connected to its four nearest neighbours on the same level (except for the elements on the perimeter), four 'children' on the next lower level (except for the elements on the base level) and to its 'parent' on the next higher level (except for the root) as in the figure 3.4. The overall number of elements in the pyramid with (N+1) levels is $4^N + 4^N/3$.

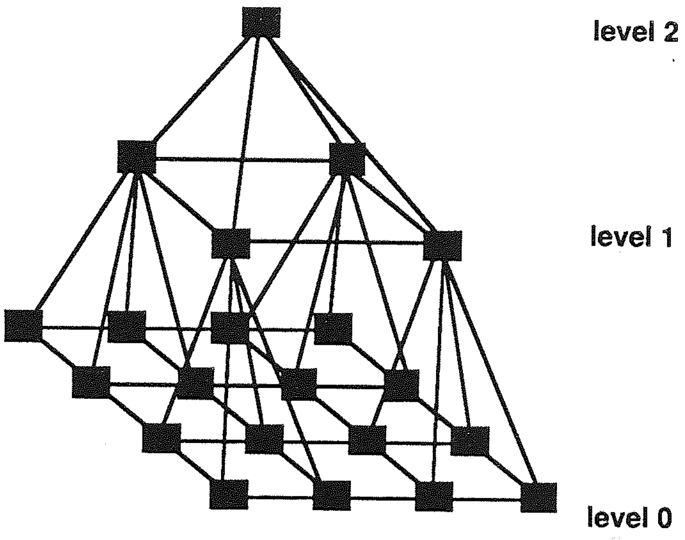


Figure 3.4 Pyramid with 3 levels.

The Pyramid has an extra 4/3 times the number of PEs required for the same size of processor array. However, the major advantage of the pyramid is the speed up in intercommunication between processors. For example, to compute a global sorting over an N by N processor array, N steps are needed, while the same computation on a pyramid needs only $\log_2 N$ steps, so there is a speed up of $N/\log_2 N$ which is high for larger systems (Stout, 1983).

There are mainly two types of pyramids depending on their granularity:

Fine granularity pyramid

In this type, each processor takes only one image pixel. A parallel access to neighbouring pixels is provided for intercommunication, so as to efficiently implement many low-level vision tasks. Prototypes have been designed for these systems, for example, PAPIA (Cantoni *et al.*, 1985) and GAM (Schaefer *et al.*, 1987).

GAM Pyramid

This is a multi-layer structure of MPP processors, adders, and sum-OR circuitry. It consists of 341 PEs with 8 Kbits for each object identification task. It is suitable for binary images. PYRASM is the GAM pyramid assembly language, a combination of microcode for the pyramid PEs and control unit commands. This mixed code format of instructions allows great flexibility. The GAM pyramid is the first five-level pyramid in operation. It has been designed and constructed at George Mason University.

Coarse granularity pyramid

In this type, each processor takes an image block which means that less data need be interchanged between the processors than in the fine granularity one. For VLSI implementation, it is too costly to implement such systems.

3.3.3 Multiresolution on Flat Arrays

This type of architecture has been proposed by Reeves (1986) to implement the fine grain pyramid algorithm on a 2-dimensional processor array (flat array). This proposed system is suitable for certain applications where the extra hardware needed for a pyramid is not justified. It is also used to evaluate the performance of the pyramid algorithms and to evaluate the benefit of extra processors required for the pyramid computer. Figure 3.5 overleaf shows the principle of mapping pyramid algorithms onto a flat array.

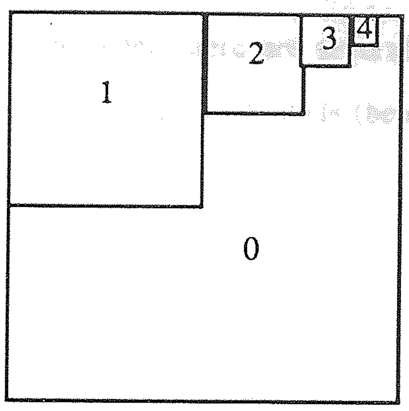


Figure 3.5 Multiresolution in flat array architecture.

3.4 Performance Evaluation

It has been shown above that there is a variety of multiprocessor architectures with different design alternatives. They are proposed, implemented, and made commercially available, but their relative merits are difficult to assess. It is vital to have methodologies and tools for predicting and evaluating their performance since they will help the designer to chose the architecture which gives the best performance for specific or versatile applications.

Performance evaluation techniques can be classified into two main areas (Marsan *et al.*, 1986), the measuring and the modeling. Figure 3.6 on the next page shows the classification tree.

Performance measure

There are three techniques for measuring the performance.

Performance measurements: These are performed where there is a real system. This gives a very accurate performance evaluations with respect to the specific system and to its work load.

Benchmarking: this is used when there are several system designs to be evaluated. In this technique, several work loads (bench marks) are used for different system design.

Prototyping: This technique is used when there is no physical computing system available. Then it is necessary to build its approximating prototype (emulator). The prototype can be used to perform measurements, possibly with bench mark programs.

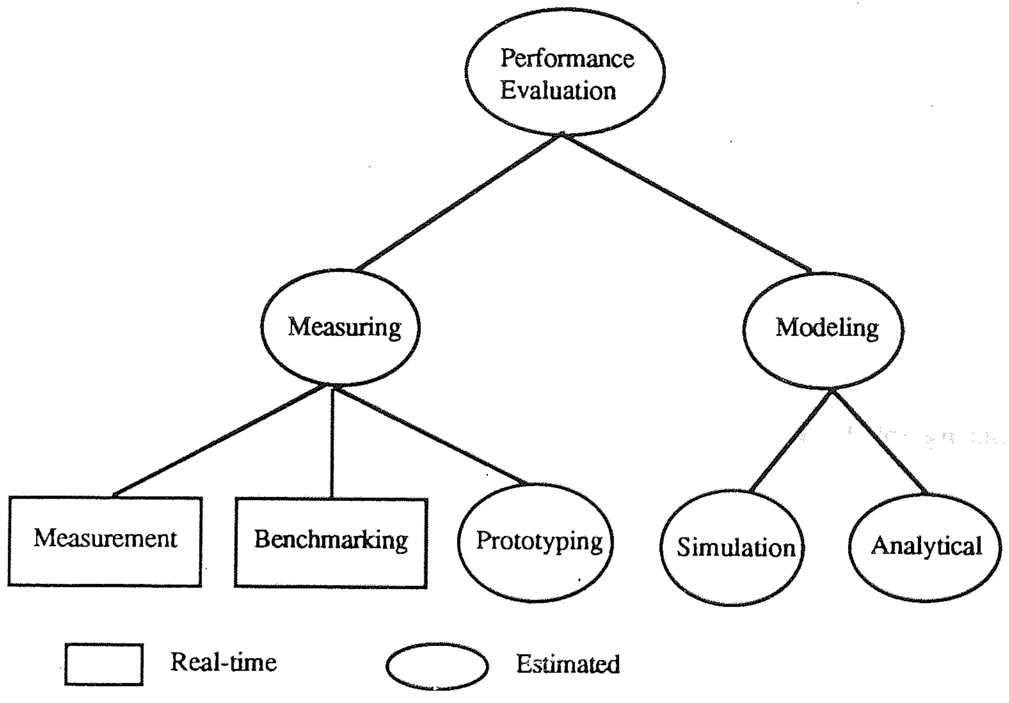


Figure 3.6 Performance classification tree.

Performance Modelling

This type of evaluation is used when there is no physical computing system available probably because the cost of building such system is very high or the available technology can not provide it. Models can be divided into two main classes:

Simulation models: they are computer programs in which system characteristics and work loads are described using appropriate algorithms.

Analytical models: They describe the system characteristics and work loads in mathematical terms. The performance can be evaluated analytically or by using numerical solutions of the resulting mathematical models.

The performance evaluation technique used here is the performance measurement one, since there is a real system under real operating conditions.

3.4.1 Performance Measurement

The performance evaluation of the parallel system is a function of several factors such as number of processors, problem size, problem type, the mode of parallelism, and the type of interconnection network.

There is no single performance measure which takes in consideration all the above factors and the overall performance of a system is normally assessed by examining individual measures. This is again taking the cost of design and construction of the system into consideration.

The performance measures are as follows, but not limited to just these.

Execution time

This is the time needed to execute an algorithm (A) for solving a problem of certain size (N) using a network of P processors. Let T(P, N, A) represent this time. The execution time used here includes distributing the data, processing the algorithm and communication. The execution time is measured from the time of starting the distribution of the data from the host processor to the network until the time of receiving the last result from the network. The number of processors in the network is indicated by P.

Speed up

This measure is for expressing how well an algorithm performs on a network of P processors as compared to one processor. Let S(P, N, A) be the speed up, so that

$$S(P, N, A) = \frac{T(1, N, A)}{T(P, N, A)},$$

where T(1, N, A) is the execution time of one processor. Usually S is between 1 to N. However, S could be more the N for some SIMD machines which use instruction queue (Kuehn & Siegel, 1986), but in this case it is called 'rough speed up' (Finkel, 1987) where the algorithm exhibits a speed up anomaly.

Efficiency

E(P, N, A) is defined as

$$E(P, N, A) = \frac{S(P, N, A)}{P}.$$

The values of E range from 0 to 1. For the rough efficiency, it should be defined analogously to the rough speed up. For example, for the SIMD machine designed by Kuehn *et al.* (1982), which uses decoupling between the control unit and the processor elements, the rough efficiency is

$$E(P, N, A) = \frac{S(P, N, A)}{2P}.$$

Useful-process point

Up(P, A) is the size N of a problem for which better speed up is achieved when using as many as P processors,

Up(P, A) = Smallest N such that

T(P, N, A) < T(P-1, N, A).

In the original definition by Finkel (1987), equality as well inequality is included in the above formula but it is not included here because there is no speed up benefit in this case.

Utilisation

U(P, N, A) measures the fraction of the time during which the processors are actually executing algorithm computations. The processors are idle when they are "busy-waiting". Siegel *et al.* (1982) suggested the calculation of this measure by counting the number of processors active for each computation step. Assume that for certain algorithm of X steps P_X is the number of processors active during step x, where $0 \leq x < X$; then the utilisation can be defined as:

$$U(P, N, A) = \sum_{x=0}^{X-1} \frac{P_x}{PX}.$$

3.5 Summary

The need to quickly process the large amounts of data present in image processing applications has stimulated the growth of parallel image processing computers. Their advances depended on the advances in technology. Since there is no single architecture suitable for *all* image processing applications a large number of different architectures have been developed. The parallel architectures may be classified on the basis of the multiplicity of instructions and data, the main types being SIMD, MIMD and MISD.

The SIMD architecture was shown to be suitable for tasks which include local operations. The MIMD architecture was shown to be suitable for tasks which include global operations. The MISD architecture was shown to be suitable for applications where the number of operations needed is not more than the number of stages of the architecture.

It seems very unlikely that any single parallel configuration will ever be optimal for *all* image processing (and other) applications. The ability to flexibly re-configure the architecture appears to be therefore of the foremost importance. Although it may be possible to implement any algorithm on any parallel configuration, however unsuitable, each algorithm is likely to map more naturally into some class of architectures than into the others. If parallel hardware is to perform a variety of tasks the best approach seems to be to have a reconfigurable network so that parallelism inherent in an algorithm can be exploited in the most suitable way.

The Transputer and the family of switchers have brought this possibility into the reach of developers. In this work an array configuration of Transputers has been used for the direct convolution and the Row -Column method of the Fast Fourier Transform, as described in chapter six. The pyramid architecture of Transputers has been used for the Vector-Radix method of the Fast Fourier Transform and for the Split and Merge algorithm, as will be seen in chapters six and eight.

The next chapter describes the Transputers and its development environment, including the programming language Occam and the proprietary Transputer Development System (TDS).

Chapter 4

Occam and Transputer

Occam is a sophisticated and powerful programming language with inherently built in concepts of concurrency and communication. The Transputer is a VLSI 32 bit processor with memory and four links through which it can communicate with other Transputers. Its features make it very suitable for building concurrent systems. The Transputer and Occam have been designed concurrently and, as such, they claim to provide an optimal solution to the parallel processing techniques. They represent a big step in the history of parallelism.

This chapter describes briefly the most important features of the Transputer, Occam and the Transputer development system.

4.1 Occam

The language Occam has been developed by INMOS (May, 1983, INMOS, 1988a) on the basis of communication and concurrency. Occam enables the system to be considered as a collection of processes running in parallel and communicating through channels. The internal contents of each process is hidden from the other processes and the only way of communication is the channel. The process can represent any application or device. Figure 4.1 overleaf shows an Occam model.

Occam is a flexible language which allows its processes and their channels to be implemented in many different ways. This allows the designer to choose the suitable technology for implementation taking into consideration the performance and cost factors.

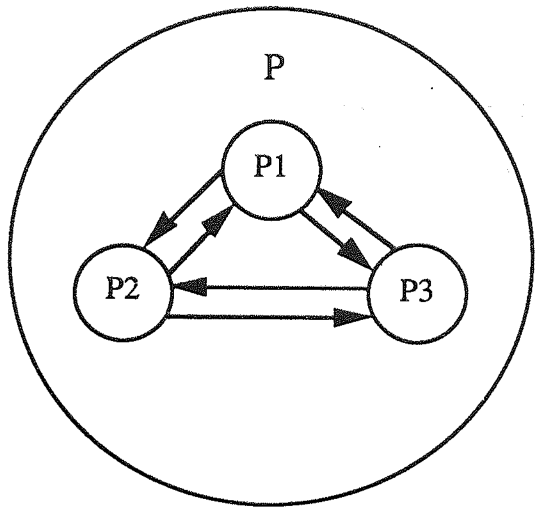


Figure 4.1 Occam model.

Occam language is designed to be implemented directly by a network of processing elements. This feature is vital for parallel processing techniques since the main objective is to get the best performance from the parallel system; this is achieved by choosing both the appropriate parallel language and the processor to implement the language efficiently. This feature is shown by the possibility of using Occam both as a high level language and as an assembly language for parallel systems; this is because there is a one-to-one relation between Occam processes and processing elements and between channels and the links between processing elements.

One other important feature of Occam is that the same parallel program can be implemented on one processor or on a network of processors. The only difference between the two is that some additional code is needed for configuration when the program is implemented on a network of computers.

4.1.1 Processes

Occam process is of finite type, i.e, it starts, performs a number of actions and then terminates. It may contain a set of processes running either in sequence or in parallel. Each process of the set may contain other sets of processes and this shows the hierarchical block-structure of Occam.

Figure 4.2 shows one Occam process containing several processes which start with process P1 and terminate at the end of the last process in the last level process P8.

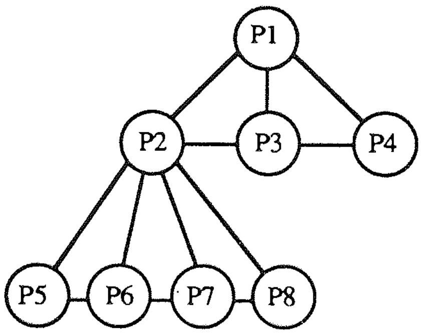


Figure 4.2 The hierarchical structure of Occam.

A very important feature of the Occam process is the locality, i. e., each process has its own variables. The importance of this feature is apparent due to the fact that the speed of communication within the chip is much faster than the speed of communication between different chips.

There are three primitive processes:

v := e	assign expression e to variable v,
c ! e	output expression e to channel c,
c ? v	input variable v from channel c.

A number of primitive processes can be combined to form more complex constructs and they are briefly presented below.

SEQ. A sequential construct can be represented by

```
SEQ
  P1
  P2
```

where P1 and P2 are the sequences of processes which are executed one after another.

PAR. A parallel construct which can be represented by

```
PAR
  P1
  P2
```

where P1 and P2 are the processes executed in parallel. The components of a parallel construct may not share access to variables, and communicate only through channels.

PRI PAR. A priority parallel which may have two components: The priority 0 (high priority) and the priority 1 (low priority). The low priority processes are executed if there are no active high priority processes.

```
PRI PAR
  P1
  P2
```

where P1 is a high priority process and P2 is a low priority process.

IF. A conditional construct

```
IF
  condition 1
    P1
  condition 2
    P2
```

where only one of the processes P1 or P2 is executed if its condition is true.

ALT. An alternative construct

```
ALT
  input 1
     P1
  input 2
     P2
```

where only one of the processes P1 or P2 is executed if its input channel is ready. It is called an input guard and it plays a very important role in the parallel program implementation.

WHILE. A loop

```
WHILE     condition
  P
```

where P is repeatedly executed as long as condition is true.

4.1.2 Synchronised Communication

The communication between two processes occurs by a channel. The channel communication is synchronised and unbuffered which greatly simplifies programming and allows for efficient implementation. When either the input channel or the output channel is ready to communicate it will wait until the other is ready.

The channel has to be named. By establishing a named channel between two processes, neither process need have any knowledge of the internal details of the other, and the internal structure of each process can change during execution of the program.

The parallel construct and the named channel features of Occam allow for decomposition of an application into a hierarchy of communicating processes. This makes Occam a convenient tool for large-scale applications.

4.2 Transputer

The Transputer is a CMOS microcomputer with on-chip RAM for high speed processing (INMOS, 1989), a configurable memory interface and four standard INMOS communication links. Figure 4.3 shows the block diagram of the Transputer.

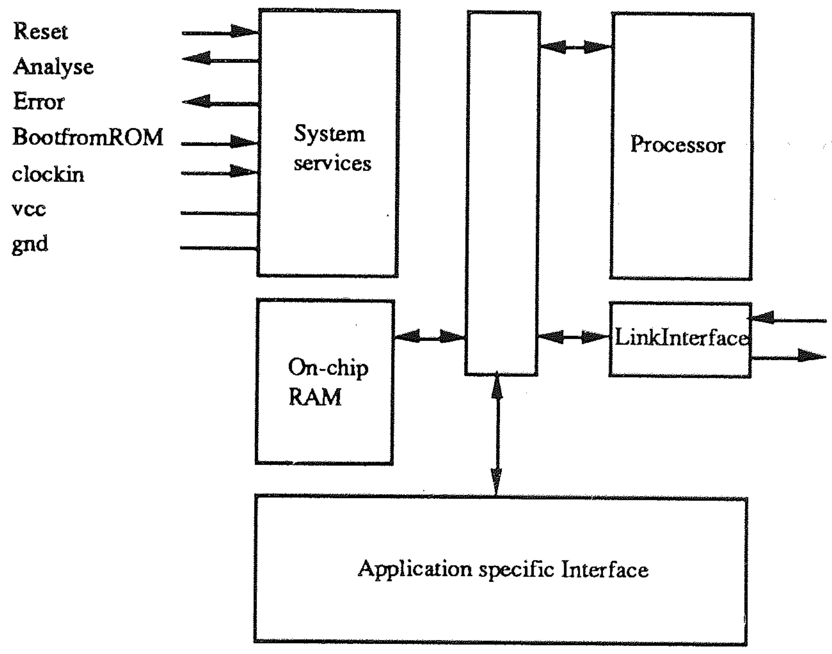


Figure 4.3 Transputer block diagram.

The Transputer is an ideal hardware architecture for implementing Occam process. Either a single process or several concurrent processes can be implemented on it. In the case of concurrent processes the concurrency will be simulated by hardware with no software intervention. The concurrent program which runs efficiently on a single Transputer can be directly implemented on a network of Transputers provided that the communication between processes is not too complicated.

The Transputer support for Occam model of concurrency is by its microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. So at any time, a concurrent process may be either active (- i.e. being executed or on a list waiting to be executed) or inactive (- i.e. ready to input; ready to output; or waiting until a specified time).

Inactive processes do not consume any processor time. The use of the scheduler removes the need for a software kernel.

The Transputer implements the Occam channel using point-to-point serial communication links. This is used to provide maximum speed with minimal wiring. This is achieved because in the case of shared bus communication, electrical problems of driving the bus require that the speed is reduced, in addition to that control logic and wiring are required to control sharing of the bus.

A Transputer link comprises input channel and output channel. A link between two Transputers is implemented by connecting the link interface of one Transputer to the link interface of the other Transputer. Thus the connection is implemented between links and not between channels. For example, consider figure 4.4 shown on the next page. The connection shown in (a) is correct as the both channels of one of the links of T1 are connected to the both channels of one of the links of T2 . The connection shown in (b) is incorrect as two channels of the same link of T1 are connected to two channels which belong to two different links of T2.

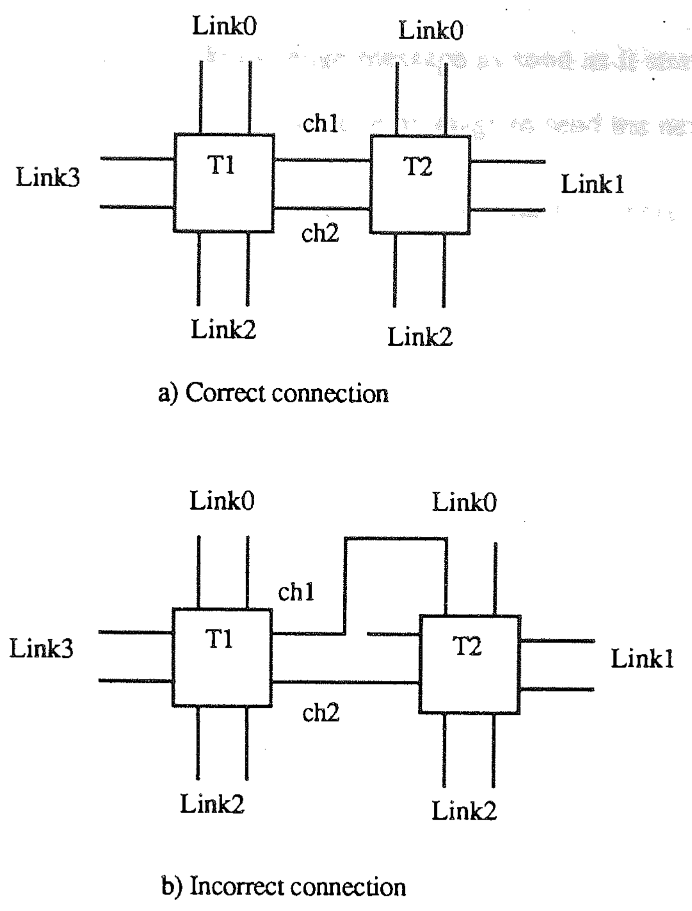


Figure 4.4 Link implementation in the Transputer.

There is a simple protocol provided in the Transputer for sending messages between channels. Messages are transmitted as sequences of bytes, each of which must be acknowledged before the next is transmitted. A byte of data is transmitted as a start bit followed by one bit followed by eight bits of data followed by a stop bit. An acknowledgment is transmitted as a start bit followed by stop bit as shown in figure 4.5.

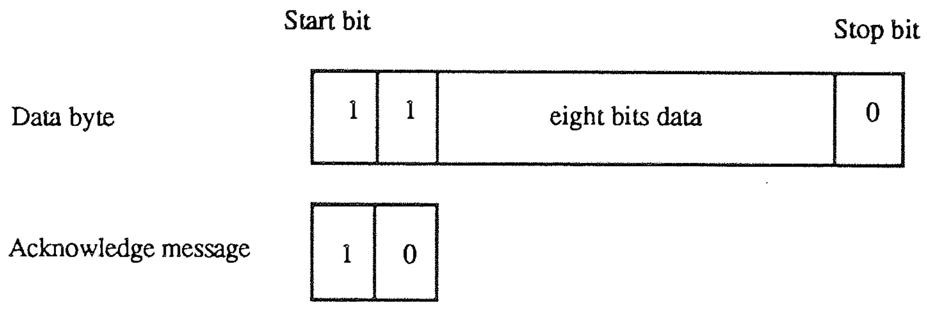


Figure 4.5 Channel protocol of the Transputer.

The receiver starts to send the acknowledge message as soon as it starts to receive the data. The transmitter waits for acknowledge message to send the next data byte.

The data and acknowledge message can be overlapped on IMS T800 Transputer, but not on IMS T414 Transputer. IMS T414 Transputer is used in this project with 15 MHZ clock rate and 10 Mbits/sec speed link.

The Direct Memory Access (DMA) is used so the links copy data from the memory of one Transputer into the memory of another with minimal software set-up costs. Since each link can transfer data in both directions simultaneously all the links and the processes can be active at the same time.

Transputer supports two levels of priority. This is achieved by the scheduler, and these levels can be specified by software.

4.3 Transputer Development System

The Transputer Development System (TDS) is a complete, self-contained programming environment developed by INMOS to support the programming of Transputer networks in Occam (INMOS, 1988b). It comprises an integrated editor, file manager, compiler, and debugging system.

The TDS runs on a Transputer board (IMS B004). This board can be installed inside an IBM PC or any of its compatibles, which provides a means of interfacing keyboard, screen and disks to the Transputer. There is a program running on the IBM PC called the 'server', which provides the TDS with access to the terminals and filing system of the IBM PC.

The TDS programmer can edit, compile and run Occam programs entirely within the development system. There are several way for loading the code to the single or a network of Transputers:

1) Occam programs can be developed on the TDS and configured to run on a network of Transputers with the code being loaded onto the network from the TDS.

2) An operating system file can be created which contains the Occam program and will boot a single Transputer or network of Transputers.

3) The TDS can be used to create Occam programs, for single Transputer or network of Transputers, that operates completely independently of the TDS. These programs are called 'stand alone' programs, and their code can be placed in EPROM. Usually they represent the final version of the design, i.e., they are tested to work perfectly by the first method.

The TDS provides all the necessary software tools and utilities to support this kind of development; such tools are the libraries for mathematical functions and input/output functions. There is a sophisticated debugging tool and software to analyse the state of a network.

4.3.1 Folding

The editor interface is based on a concept called 'folding'. The folding operations allow the text currently being entered to be given a hierarchical structure which reflects the structure of the program under development. The fold could be filed, and the filed fold can contain several filed folds as follows:

```
{{{F example
...F  P1
...F  P2
SEQ
      P3
      P4
}}}
```

In this example the filed fold 'example' contains two filed folds called P1 and P2.

4.3.2 Compiler

For any Occam program to be compiled, two conditions must have been met. First, the fold containing the source must be filed. Second, this source fold must be enclosed by a 'compilation fold', to which the compiler will be applied.

There are five types of compilation folds;

EXE- It is an 'executable' program designed to run within the TDS. It can access channels which communicates with the screen, keyboard, and filing system.

UTIL - It is a utility set within the TDS. It has more complex environment than an EXE. The utility interfaces are currently not available to the TDS users.

PROGRAM - It is an Occam program which intended to run on a network of Transputers. It contains configuration information that enables the development system to load the program into a Transputer network. The PROGRAM can not run within the TDS.

SC - It is a separate compilation unit. SC is not a complete program and it is usually contained within another compilation fold.

LIB - It is a library compilation unit. It contains a number of constants, procedures, and function declarations that may be shared between parts of a program or between different programs.

4.4 Summary

This chapter, which has described Transputer hardware, Occam programming language and the Transputer Development System, completes the introductory part of the thesis. The chapters that follow will describe the design of the overall system architecture, used for all the implemented algorithms; and the design, implementation and analysis of the selected segmentation algorithms. All the work described below has been developed using Transputers and Transputer-related hardware and all the program code has been written in Occam.

Chapter 5

System Architecture Design

The development of parallel processing applications involves not only the design of software algorithms but, unlike on Von Neumann machines, also the design of the overall 'system architecture', with elements such as the number of processors, location and size of memory modules, communication channels and their interconnections. It was quite natural to split the design into two parts: system level design and algorithm level design. This chapter is concerned with the system level design and particularly with the flexible interconnections of input and output channels, intercommunication protocols and message routing procedures. The actual design relies strongly on the property of Occam which allows one to model in software several concurrent processes on a single processing element.

5.1 Software and Hardware Architecture set-up

Throughout this project all the developments have been carried out using the same overall software and hardware set up. The Transputer system has been configured as shown in figure 5.1.

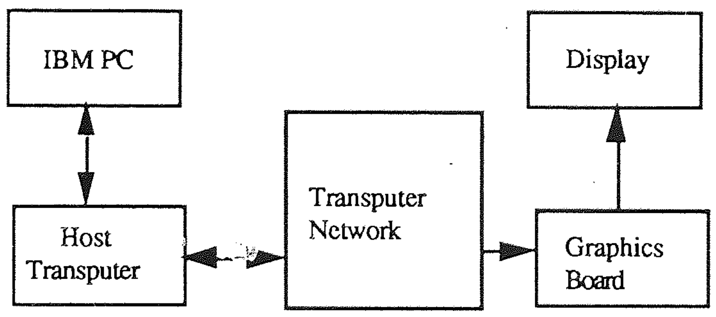


Figure 5.1 Transputer system for image processing.

The details of the elements of this hardware configuration together with corresponding software structure developed for this project are described in the sections 5.2-5.4 below.

5.2 Host Transputer

The host Transputer is responsible for interfacing between the IBM PC terminals and disks and the network of Transputers. The Occam model of the process used for this Transputer is shown in figure 5.2. It is an EXE compilation fold with four processes running in parallel and sharing the time of the processor. The complete program code is in Appendix B.

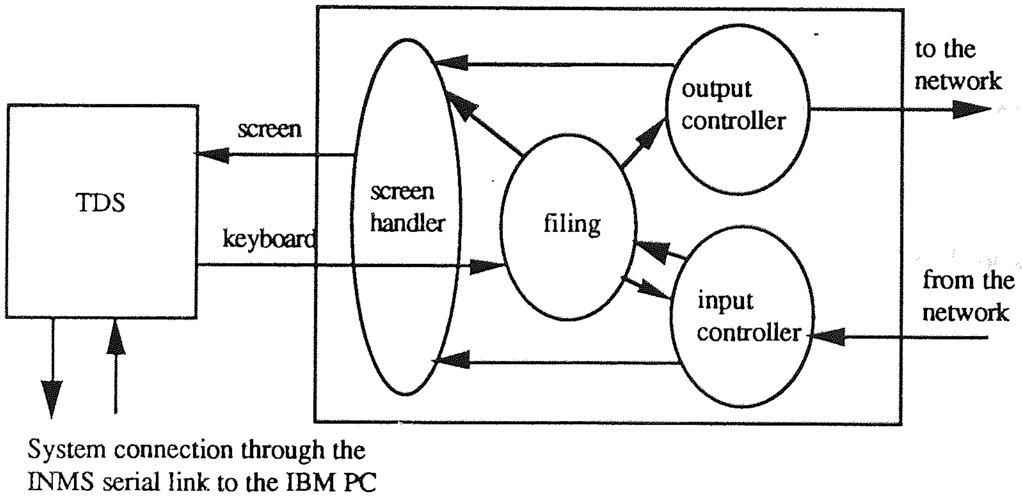


Figure 5.2 Occam model for the host Transputer.

Screen handler

The screen handler is repeatedly searching for one of four alternative channel inputs: the input controller channel (controller.in.screen), the output controller channel (controller.out.screen), the filing channel (filing.screen), and the echo channel. For each alternative a suitable action is taking place using suitable libraries supplied by the software of the TDS. The main body of the procedure is as follows:

```
WHILE TRUE
    ALT
        controller.in.screen ? CASE
            action.1
        controller.out.screen ? CASE
            action.2
        filing.screen ? CASE
            action.3
        echo ? CASE
            action.4
```

Filing

This represents the main process of the EXE program which is responsible for the input of data from the filing system, sending the data to the controller output, receiving data from the controller input and sending the result back to the filing system if necessary. It also calculates the time spent on processing. The keyboard handler is also included within this process.

For accessing the filing system several procedures are written for reading and writing files on the IBM PC filing systems.

Output Controller

It is like a buffer for sending the data to the network.

Input Controller

It is like a buffer for input of the results from the network and it is responsible for making sure that all the results have come from the all Transputers in the network.

5.3 Network of Transputers

Transtech TSMB-16 (Transtech, 1988) Transputer motherboard is used to host a network of Transputers. This motherboard for the Transtech TSM42/82 Transputer modules is capable of holding up to 16 modules mounted vertically. It contains two INMOS C004 programmable link switches for reconfiguring the network of Transputers and two intelligent RS-232 serial ports for connection with other microprocessor systems. The type of the Transputer used is IMS T414 with 15 MHZ clock rate and 10 Mbits/sec speed link.

Programmable Link Switch

IMS C004 is a transparent programmable link switch designed to provide a full crossbar switch between 32 links inputs and 32 link outputs. The switch is programmed via a separate serial link called the configuration link. The block diagram of IMS C004 is shown in figure 5.3 overleaf.

A configuration message consists of either one, two, or three bytes. It is received via the configuration link. IMS C004 configuration messages (INMOS, 1989) are summarised in Table 5.1.

Configuration Message	Function
[0] [input] [output]	Connects input channel to output channel.
[1] [link1] [link2]	Connects link1 to link2.
[2] [output]	Enquire which input the output is connected.
[3]	This command must be sent at the end of every configuration sequence.
[4]	Reset the C004 switch.
[5] [output]	Output is disconnected.
[6] [link1] [link2]	Disconnect outputs of link1 and link2.

Table 5.1 IMS C004 configuration messages.

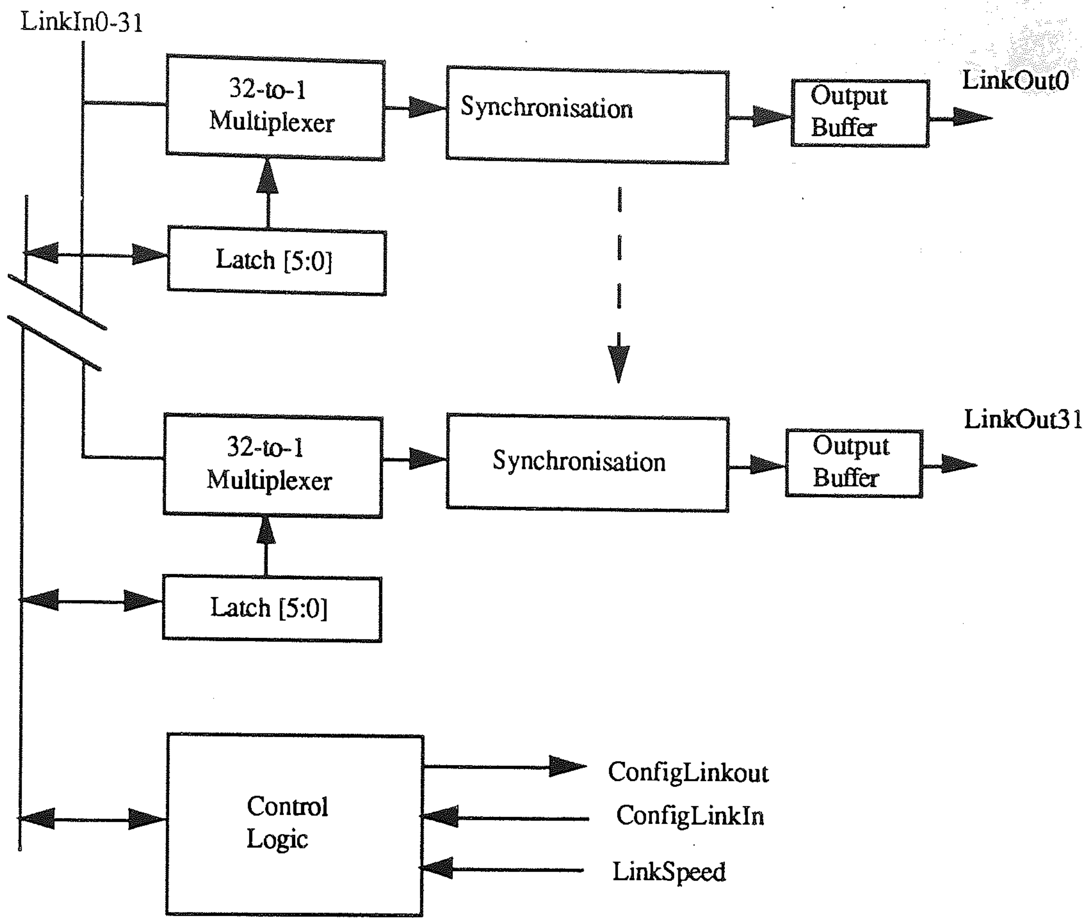


Figure 5.3 IMS C004 block diagram.

LinkIn0-31 are the link inputs of the Transputers.

LinkOut0-31 are the link outputs of the Transputers.

For line configuration with more than 32 links, several switches could be used since they can be cascaded to any depth without loss of signal integrity.

Serial Ports

The motherboard has two intelligent RS-232 serial ports for connection with other microprocessor systems. Data on link can be converted to RS-232 and vice versa by INMOS C012 link adaptors.

Transputer Moterboard

The block diagram of the TSMB-16 Transputer board is shown in figure 5.4.

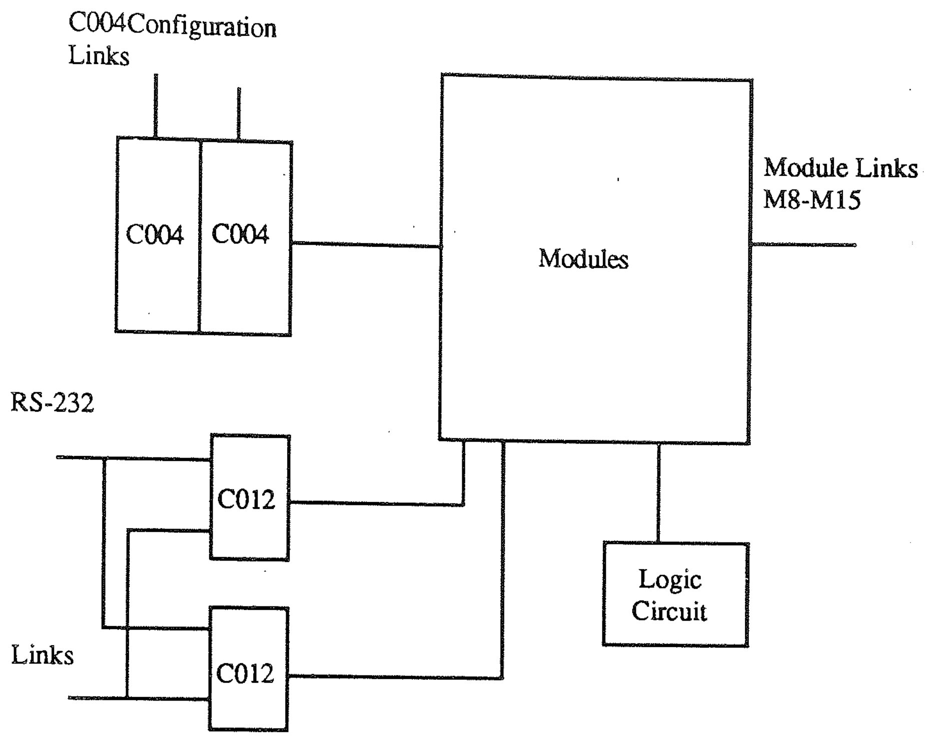


Figure 5.4 TSMB-16 block diagram.

Link 3 of a group of eight modules are connected to the C004 via switches, so that they can be used for external connections if required.

Links 0 and 1 of modules are connected to C004-1 and links 2 and 3 are connected to C004-2. According to this connection:

1) Link 0 of any module can be connected via C004-1 to link 0 of the same module or to link 0 or 1 of other modules.

2) Link 1 of any module can be connected via C004-1 to link1 of the same module or to link 1 or 0 of other modules.

3) Link 2 of any module can be connected via C004-2 to link2 of the same module or to link 2 or 3 of other modules.

4) Link 3 of any module can be connected via C004-2 to link3 of the same module or to link 3 or 2 of other modules.

This gives the flexibility for configuring a wide range of topologies.

5.4 Configuring the network

The Host Transputer is responsible for configuring the network. Configuration proceeds in two phases: 'hardwired' configuration and software configuration. The 'hardwired' configuration will not be changed during the execution of an Occam program. It is achieved by configuring the two C004 switches by software to create a 'hardwired' connection.

The software for configuring the two C004 switches has been designed and written as a part of the project. It is a general procedure for connecting two links. It is versatile and easy to use. There are four entries to that procedure: numbers of two links to be connected and their module numbers (each module is specified by an identification number). The programmer needs only to alter these numbers for various configurations. A program is also written for configuring the network for various specific topologies such as array, tree, and others.

The second step is the software configuration which is the allocation of addresses to the channels of the network of Transputers. This is achieved by using the PLACE command of Occam which allocates a variable, a channel, a timer, or an array at an absolute location in memory. Prior to this an allocation process must allocate an element to a compatible location. A channel should be placed at a location which implements a channel.

For each Transputer in the network there are eight channels. They are allocated in a certain addresses in memory (INMOS, 1988b) and should be allocated in these locations, as follows:

```
VAL link0out AT 0:
VAL link1out AT 1:
VAL link2out AT 2:
VAL link3out AT 3:
VAL link0in AT 4:
VAL link1in AT 5:
VAL link2in AT 6:
VAL link3in AT 7:
```

For a network of Transputers which are connected in a certain topology, any channel between any two Transputers should be identified by both of them with the same name. This name should be then placed at the local location of each Transputer. For example, suppose that a name channel1 is defined for a channel which is between Transputer T1 link0in and T2 link0out; the place allocation is as follows using PROCESSOR command of Occam:

```
PROCESSOR T1 T4
  PLACE channel1 AT link0in:
PROCESSOR T2 T4
  PLACE channel1 AT link0out:
```

This can be done for all channels in the network. For a very large network, an array of channels should be defined to simplify the configuration so the PLACE PAR command could be used.

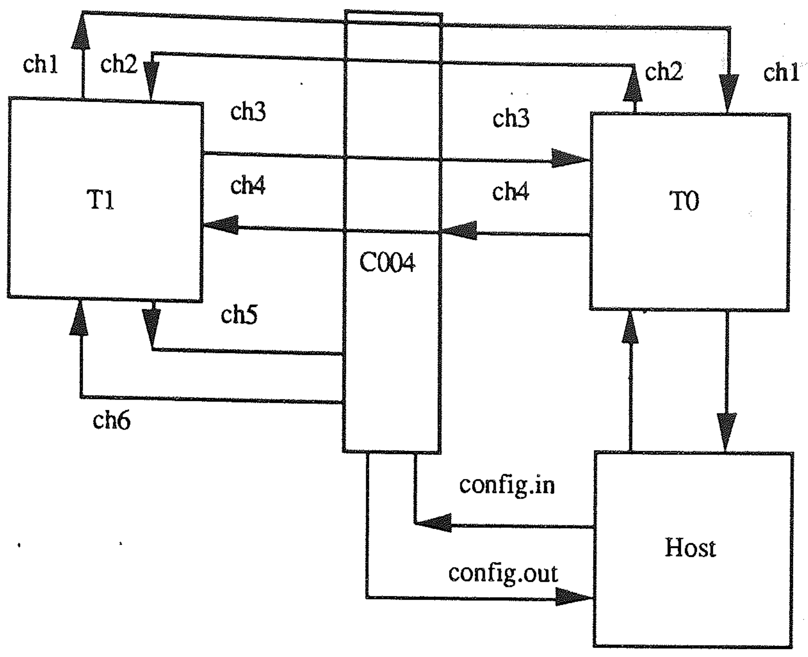
The first step is to configure the system in a certain topology. This can be achieved by the software for configuring the C004 switch.

The next step is to design the PROGRAM with the configuration requirements which match the connection between the Transputers. There is a restriction of naming the channel between two links. This condition says that the channel name should be the same on the two Transputers. This is necessary at the configuration level since it enables the development system to load the code to the target Transputers.

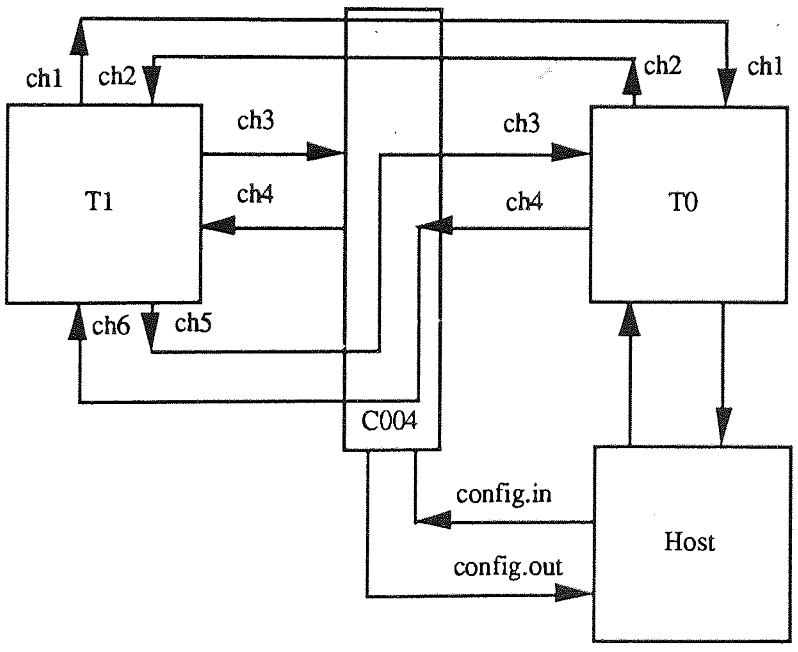
For running the program, the code should be loaded to the network. After this stage each Transputer has its code and will proceed to run the code. During the processing, a message could be sent to the C004 switch to change the connection. Here the condition that same name should be given to the channel is not valid any more because at this stage each Transputer is running separately from the development system.

Figure 5.5 on the next page shows a network consisting of the host Transputer, the C004 switch, and two Transputers.

In the configuration of figure 5.5 (a), the same name is given to the same channel connected between two Transputers. In the second configuration figure 5.5 (b), which has been implemented during the run time processing, different names are given to the same channel (for example, ch3 is connected to ch5 and ch4 is connected to ch6).



(a)



(b)

Figure 5.5 Example of reconfiguration.

5.5 Data Routing

This section describes the general data routing strategy used in this project, which includes all the data movement between The Transputers. For data routing through the network an Occam model in each Transputer in the network has been designed for efficient routing and this is shown in figure 5.6.

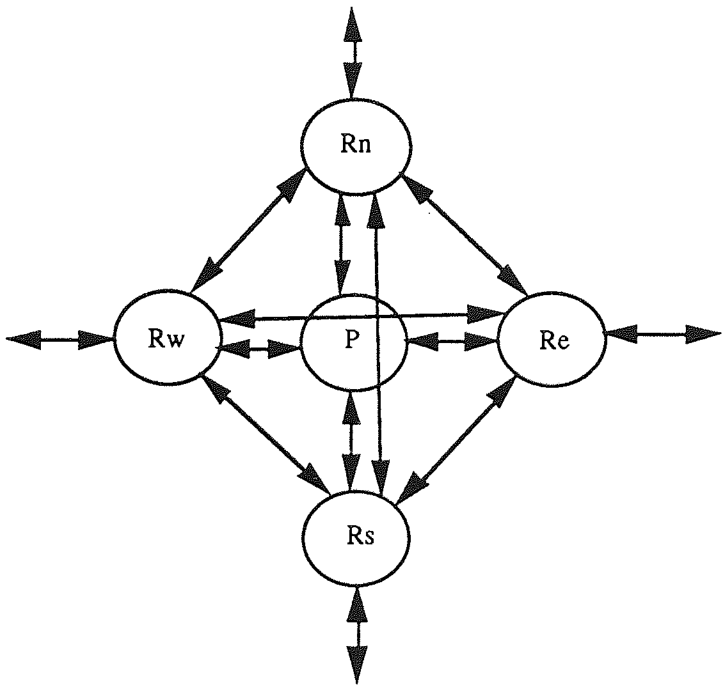


Figure 5.6 Occam model for the Transputers in the network.

Process P contains a certain application routine. Processes Rn, Re, Rs, and Rw are called routers. These routers are responsible for handling messages from other Transputers. The router checks the destination of each arriving message (the destination is always included as a part of a message). A message arriving to its destination is sent by the router to the P process; otherwise it is sent to other

Transputers according to a certain data routing layout. All these processes run in parallel. Each router contains two channels from two different links as shown in figure 5.7.

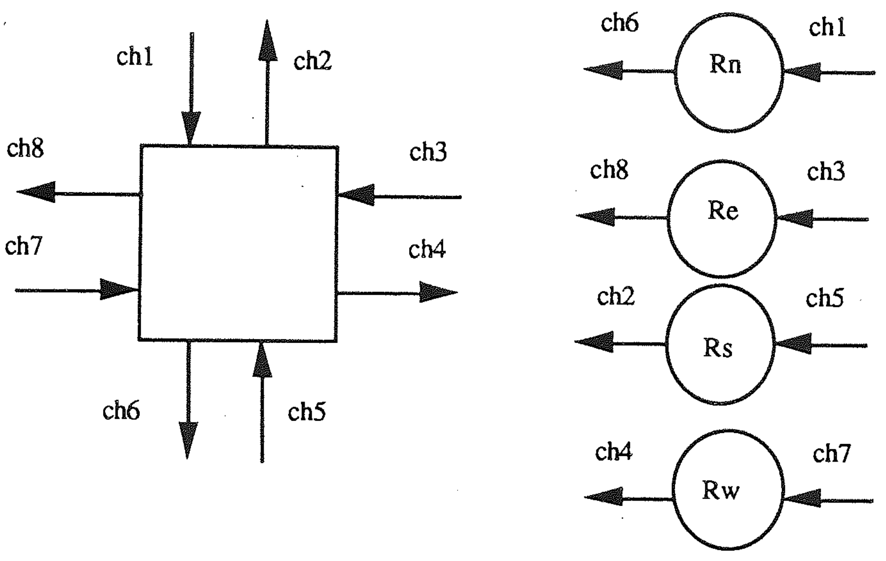


Figure 5.7 Channels allocations to the router processes.

This model has been used to provide a compromise between the problem of memory requirements and speed. In the case of minimum memory requirement it is better to use the model with a single process, however this will slow the speed of processing. On the other hand, for the case of maximum speed, it is better to use the model shown in figure 5.8 overleaf.

In this case the memory requirement is higher because each data structure of an application is declared for the process and for each router.

A protocol has been defined for each type of a channel used in the network. The protocols are stored in one library which can be accessed by all the procedures written for the network.

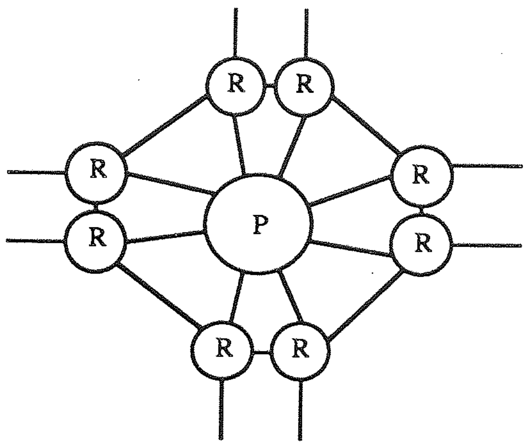


Figure 5.8 Occam model for maximum speed data routing.

There are two types of protocol: sequential protocol and variant protocol. The sequential protocol specifies a protocol for communication which consists of a sequence of data. The following example may be considered.

PROTOCOL *type1* IS INT; REAL32 :

This gives the description of protocol ***type1***, and can be assigned to any channel. This protocol means that the message sent contains two data values, the first is of type **INT** (integer) followed by single data of type **REAL32** (real value of 32 bits width). For example;

CHAN OF *type1 channel1*:

This shows that ***channel1*** is of type ***type1*** .

The variant protocol is used when a single channel is used to communicate messages with different formats. A variant protocol specifies a number of possible formats for communication on a single channel.

There are several tags for each variant protocol. These tags are used to distinguish between different protocols within one type of channel. The example of such a channel type is as follows:

```
PROTOCOL type2
    CASE
        x; INT; INT::  []BYTE
        y; REAL64; INT::  []BYTE
        z; INT; INT::  []REAL32

:
```

The tags in this type are **x**, **y**, and **z**. The tag is followed by the sequence of data. The **x** tag, for example, is followed by three fields; an integer, followed by an array of bytes of size of the value of the integer in field 3 in the protocol.

The router procedures are intelligent: they have the ability to route the message around the network speedily and efficiently.

A map of the shortest path between any Transputer and the others is designed and included in these procedures. An example of that is shown in figure 5.9.

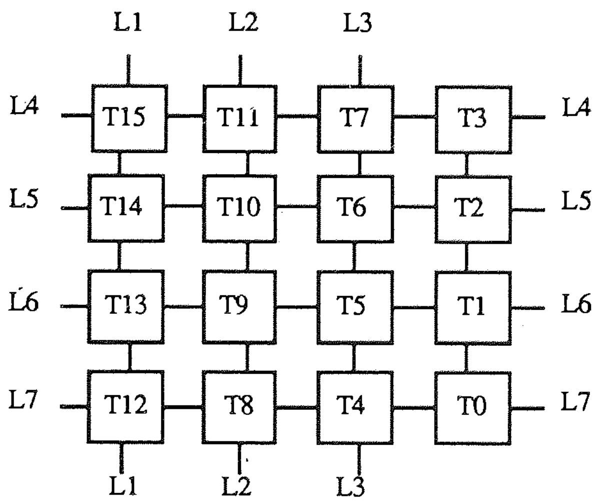


Figure 5.9 Array of size sixteen of Transputers.

For example, the shortest path between T11 and T12 is the path T11 --- T8 --- T12.

The procedure is designed for general data routing so it can be used for any application using that specific configuration. For other configurations, the modification is straightforward. This is due to efficiency of the program design of the router procedure.

5.6 Summary

On the system and network level all processes use the same hardware and communication model. This means that the same set of procedures can be used for common operations such as message routing or timing. A model chosen for message routing consists of the processing module and four router modules. The advantage of this model is that these two different types of activities, data processing and data flow, can be developed independently. This leads to overall clearer software logic. Another advantage is that many algorithms can use the same routing scheme and a code for this does not need to be embedded in a data processing code.

The next three chapters describe configurations and software which are specific to given algorithms. It should be kept in mind, however, that all the message routing and timing operations are performed using the general hardware and software model described in this chapter.

Chapter 6

Convolution: Theory and Implementation

Chapter two identified the convolution operation as one of the most common operations in image processing. It is well known that the convolution of two image functions in the spatial domain corresponds to the multiplication of their respective Fourier transforms in the frequency domain. The choice of the particular domain depends mainly on the size of the convolution kernel. For large images or large kernels it becomes computationaly more effective to perform the convolution in the frequency domain (Dudgeon & Mersereau, 1984). In the course of this work both methods, spatial and frequency, have been implemented. The details of the spatial domain (direct) convolution are described first, in section 6.3.

As the Fourier transform is a crucial part of the frequency domain convolution a lot of effort has been put into its implementation and analysis. Two methods of 2-dimensional Fast Fourier Transform have been implemented: the Row-Column method and the Vector-Radix method; each of these methods offers different advantages in different algorithmic context. Their implementation and evaluation are described in detail in sections 6.5 and 6.6 respectively. Section 6.7 describes implementation and analysis of convolution in frequency domain through both methods the Row-Column and the Vector-Radix. All the convolution methods are then compared in section 6.8.

6.1 Convolution

There are two types of convolution: circular (cyclic) convolution and linear convolution. This section outlines their theoretical basis.

6.1.1 Circular Convolution

For this type, suppose that there are two finite-extent sequences, x(m, n) and h(m, n), with region of support R_{MN}. Then the circular convolution is the operation of circularly shifting h past x. This can be illustrated as follows:

Let y(m, n) be the output of the circular convolution, then

$$y(m, n) = x(m, n) * h(m, n), \qquad m, n \in R_{MN} \tag{6.1}$$

where * denotes the operation of circular convolution. Equation (6.1) can be evaluated as follows,

$$y(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} h(k, l)\, x(m-k, n-l). \tag{6.2}$$

The circular convolution can be evaluated by FFT techniques. Since the convolution operation in the spatial domain is equivalent to the multiplication operation in frequency domain,

$$Y(\omega_1, \omega_2) = X(\omega_1, \omega_2)\, H(\omega_1, \omega_2), \tag{6.3}$$

where Y, X, H are the FFT of y, x, and h respectively, then

$$y(m, n) = FFT^{-1}\, Y(\omega_1, \omega_2).$$

6.1.2 Linear Convolution

Linear Convolution is a special case of circular convolution in which the region of support of h is smaller than that of x, i. e. in this type of convolution h is linearly shifted past x. The convolution equation is

$$y(m, n) = x(m, n) * h(k, l), \qquad \begin{pmatrix} m, n \in R_{MN} \\ k, l \in R_{KL} \end{pmatrix} \tag{6.4}$$

Linear convolution can be implemented by two methods, the direct and the FFT methods.

Direct method

This method is based on implementing the linear convolution in the spatial domain, i.e., applying equation (6.4). For each point of y(m, n), the window h is multiplied point by point by a window of the same size centred at x(m, n) and the sum is placed at y(m, n). Figure 6.1 shows the implementation of equation (6.4).

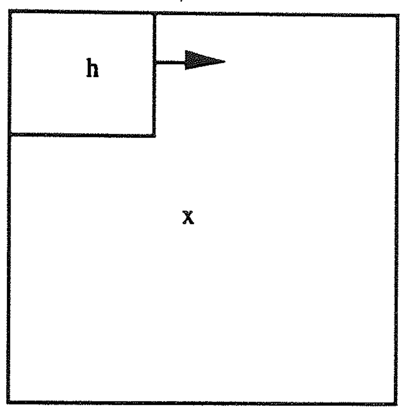


Figure 6.1 The convolution operation.

The number of multiplications needed in this method is Cd, where

$$Cd = MNKL. \tag{6.5}$$

Frequency method

In this method the results are calculated in the frequency domain by applying equation 6.3. The main difference between this method and the circular convolution method is that the size of the window and the input image is different and the FFT method requires that both sizes should be equal. This problem is solved by applying the following procedure:

1) Choose region of support R_{MN} so that

$$M \geq m + k - 1, \text{ and } N \geq n + l - 1. \tag{6.6}$$

2) Augment each x(m, n) and h(m, n) with sufficient samples of zero values to fill the region R_{MN}.

3) Compute (MN)-points FFT of both x(m, n) and h(m, n).

4) Compute $Y(\omega_1, \omega_2)$ using equation (6.3).

5) Compute the (MN)-points FFT^{-1} of $Y(\omega_1, \omega_2)$ to get the result y(m, n).

The number of multiplications needed for this method depends on the technique used for computing the 2D FFT. There are two techniques for computing the 2D FFT, the Row-Column (Dudgeon & Mersereau, 1984) and the Vector-Radix (Rivard, 1977; Arambepola, 1980)).

The main difference between the direct and the frequency method is that for the direct method the computation complexity depends directly on the size of the window h, while in the frequency method it is independent on the window size since it uses a fixed size which satisfies equation (6.6).

6.2 Parallel Architectures for Convolution

Different convolution algorithms may require different parallel architectures for efficient implementation. The array topology is quite suitable for implementing the direct method. There are two types of topologies which may be used for each technique of implementing the Fourier Transform algorithm. The array topology is suitable for implementing Row-Column technique since the matrix transposition algorithm, which is the main time consuming part of the method, maps very well into it. For the implementation of the Fourier Transform using the Vector-Radix technique the pyramid topology is a good choice because by using it the intercommunication between the processors within each level of the pyramid is not needed.

At the end of this chapter there will be a discussion of the results of the various implementation topologies.

6.3 Implementation of the Direct Method

The parallel implementation for the direct method is based on the SIMD parallel architecture. The array topology is used for the network of Transputers. The image parallelism technique is used for the implementation. Figure 6.2 shows the block diagram of the system used for the direct implementation.

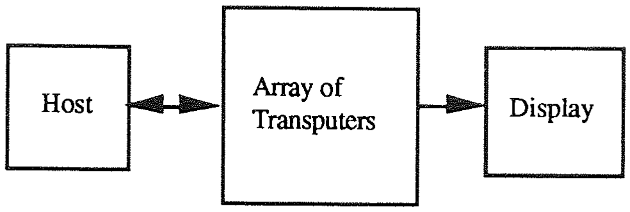


Figure 6.2 System block diagram of the the implementation of the direct method convolution implementation.

The host Transputer acts as a controller and fetches the input image from the filing system. Then it divides it into a number of subimages, the size of each depending on the size of the input image, the number of Transputers in the array, and the window size. The shape of the array need not be square, so the number of Transputers in the array could be nearly arbitrary. The main requirement is to divide the input image into subimages of equal sizes. This is necessary for load balancing between the Transputers.

Suppose that the input image size is N^2, the kernel size is K^2, and the number of Transputers is T^2. Then the number of subimages is T^2 and the size of each is B^2 where

$$B = \frac{N}{T} + K - 1 ,$$

where K-1 represents the overlap between subimages as shown in figure 6.3 below. The overlap is necessary so there will be no need for intercommunication between Transputers, which would slow the speed of processing.

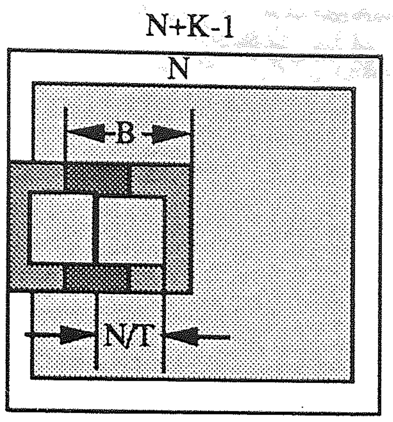


Figure 6.3 The overlap between subimages in the direct method.

The output image will be shrunk by K-1 in each dimension. To get the same output size as that of the input, the input image should be expanded by K-1 by filling the extra pixels with zero values.

Each Transputer applies the convolution operation of equation 6.4 to its subimage and then sends the result back to the host which collects the whole subimage results and combines them into one output image.

A program written in Occam has been designed and distributed to the network of Transputers. The strategy of data routing is the same as described in section 5.5.

A timer of the host Transputer is used to measure the time of processing of the direct method algorithm. This time is measured from the instant of starting the distribution of the subimages to the instant of receiving the last pixel of the last subimage result. Several image sizes with several window sizes have been tested for arrays of single, four and sixteen Transputers. Tables 6.1(a)-(c) overleaf show the time of processing with respect to image size for several window sizes for the three array sizes. The type of data is real and the type of operation is floating point. Time is given in seconds.

image side	kernel side						
	3	5	7	9	11	13	15
16	0.083	0.198	0.366	0.588	0.861	1.183	1.552
32	0.331	0.793	1.48	2.387	3.514	4.852	6.401
64	1.319	3.17	5.936	9.618	14.196	19.656	26
128	5.27	12.665	23.732	38.492	56.91	78.98	104.698
256	23.603	51.41	92.15	124.424	174.657	228.23	288.532

(a) single Transputer.

image side	kernel side						
	3	5	7	9	11	13	15
16	0.026	0.057	0.101	0.159	0.23	0.314	0.41
32	0.099	0.218	0.393	0.624	0.91	1.249	1.641
64	0.387	0.856	1.553	2.481	3.632	5.005	6.599
128	1.534	3.349	6.172	9.875	14.492	20.023	26.467
256	6.11	13.519	24.602	39.383	57.82	79.928	105.719

(b) Four Transputers.

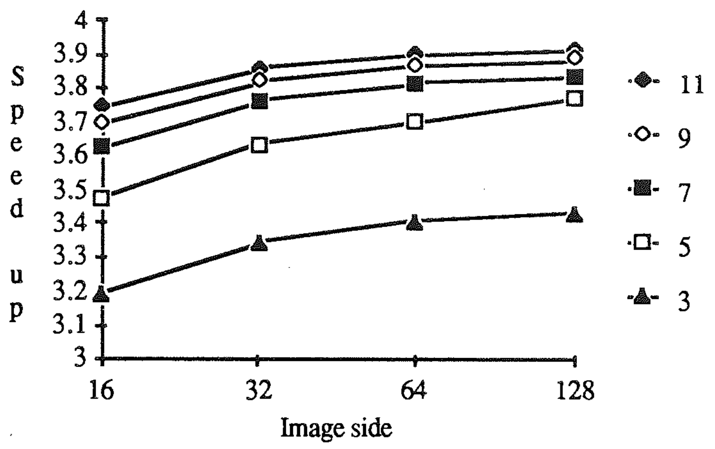
image side	kernel side						
	3	5	7	9	11	13	15
16	0.014	0.025	0.041	0.062	0.086	0.115	0.149
32	0.041	0.077	0.127	0.192	0.271	0.365	0.477
64	0.153	0.276	0.458	0.7	0.999	1.355	1.77
128	0.595	1.068	1.775	2.718	3.894	5.304	6.932
256	2.36	4.223	7.012	10.732	15.376	20.952	27.459

(c) sixteen Transputers.

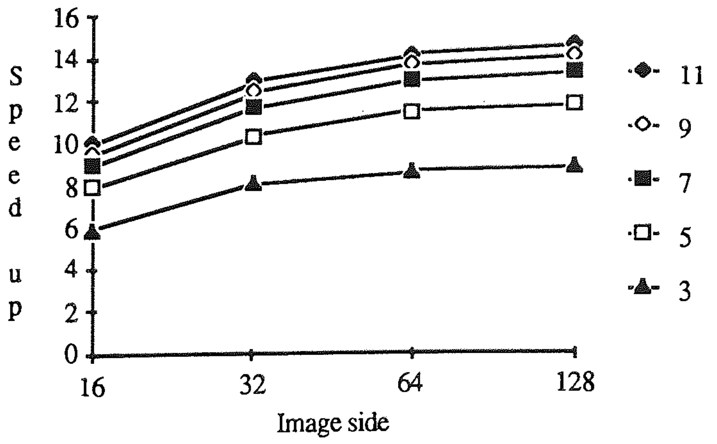
Table 6.1 Time of processing of the direct convolution for several image sizes with several kernel sizes using three array sizes.

Time is given in secs.

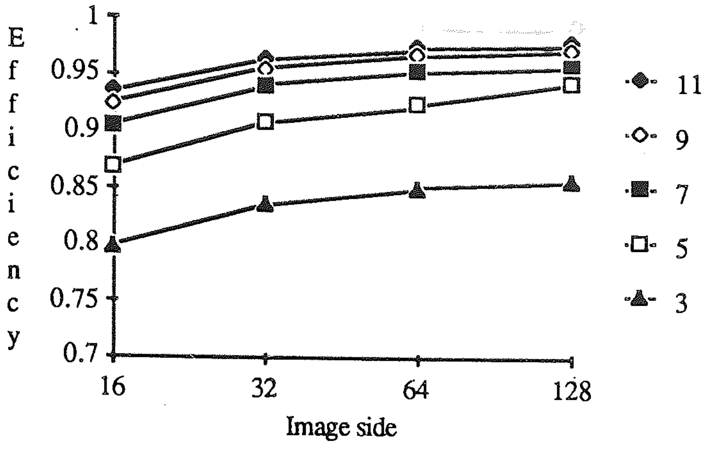
To evaluate the performance of this implementation the speed up and the efficiency (see section 3.4.1) have been calculated for each case. Figure 6.4 (a)-(f) shows the performance graphs.



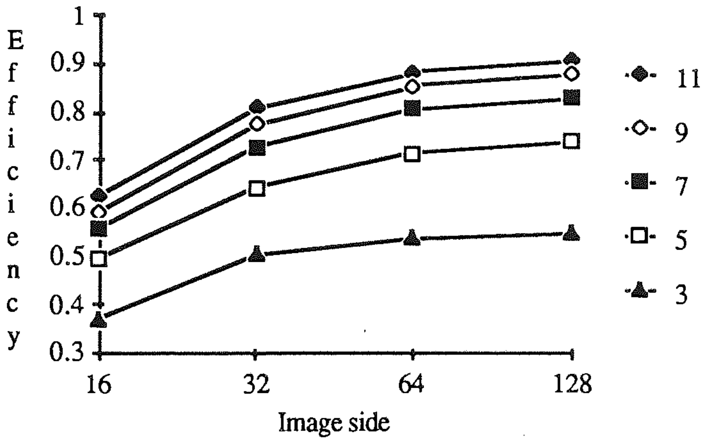
(a) Speed up versus image size for Transputer array of size 4.



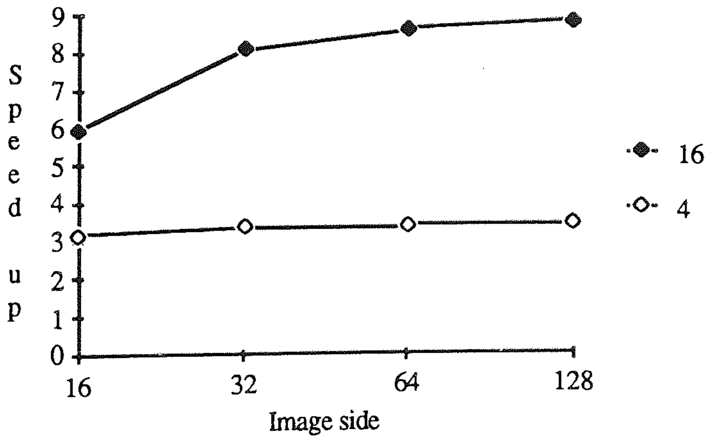
(b) Speed up versus image size for Transputer array of size 16.



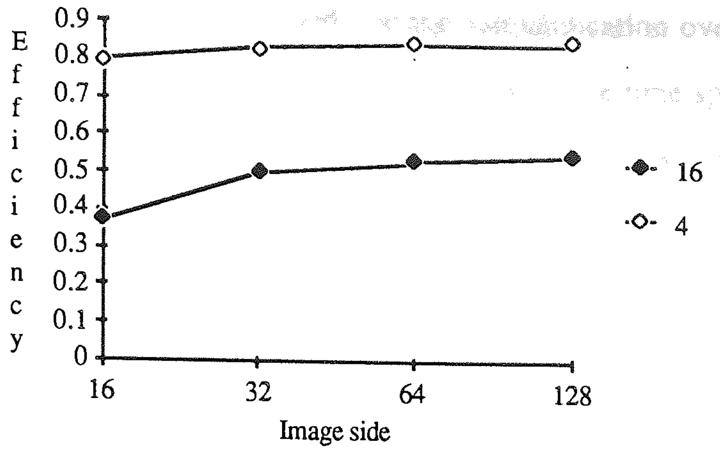
(c) Efficiency versus image size for Transputer array of size 4.



(d) Efficiency versus image size for Transputer array of size 16.



(e) Speed up versus image size for kernel size 3x3.



(f) Efficiency versus image size for kernel size 3x3.

Figure 6.4 Performance graphs for the implementation of the direct convolution for Transputer arrays of size four and sixteen.

Two sets of performance graphs were shown in figure 6.4. In the first set of graphs (figure 6.4 a, b, c, & d), two effects are shown. The first is the effect of increasing image size on the performance. It can be seen that the speed up and the efficiency increases with increasing image size. This result is quite useful since the need for parallel system is more vital for larger images, where time of processing is relatively high. The second effect is that of increasing the window size (kernel size) on the speed up and the efficiency. It can be seen that they both increase with the increase of the kernel size. This is because for larger kernel sizes the processing is more intensive than that for smaller ones. Graphs of figure 6.4 (e & f) show the effect of increasing the size of the array of Transputers on the performance of the system. It can be seen that for larger array size the speed up is higher but the efficiency is lower than that for smaller array sizes. The gap between the values of efficiency decreases for larger images which is quite useful since the parallel implementation is more applicable to larger images. The reason for the large efficiency difference in the case of smaller images is due to the proportion of time spent on data movements and processing, especially for the larger Transputer array. In other words if it is assumed that the communication overhead in this system is the ratio of time spent for data movements to the time spent for actual processing then its value is higher for lower image sizes than that for larger image sizes.

6.4 Two Dimensional Fast Fourier Transform

Two Dimensional Fast Fourier Transform (2D FFT) is a very powerful tool used in a number of applications such as image processing, seismic signals, radar detection, computed tomography, nuclear magnetic resonance (NMR) tomography etc (Gonzalez & Wintz, 1988). There exist a great deal of 1-dimensional FFT implementations on parallel machines; for a good survey see Hockney and Jesshope (1988). In contrast, relatively few parallel implementations of 2-dimensional FFT have been reported up to date (Siegel, 1981, Lui, 1983).

The discrete 2-dimensional Fourier transform can be defined as follows

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1}\left[\sum_{n_2=0}^{N_2-1} x(n_1,n_2)\, W_{N_2}{}^{n_2 k_2}\right] W_{N_2}{}^{n_1 k_1} \qquad (6.7)$$

for $0 < k_1 < N_1-1$

$0 < k_2 < N_2-1$

where $x(n_1,n_2)$ is the data in the spatial domain, $X(k_1,k_2)$ is the data in the frequency domain, and

$$W_N = \exp\left(\frac{-j2\Pi}{N}\right) \qquad j = \sqrt{-1}.$$

In this thesis, it is assumed that $N_1 = N_2 = N$ and $N=2^n$.

The direct calculation of 2D FFT requires N^4 complex additions and N^4 complex multiplications, i.e, the computational complexity is of the order O (N^4). The 2D

FFT is usually implemented by either the Row-Column method or the Vector-Radix method (Rivard, 1977). The calculations by the Row-Column method are of the order O ($N^2 \log N$), while for the Vector-Radix method they are of the order O ($\frac{3}{4}N^2 \log N$) (Dudgeon & Mersereau, 1984).

6.5 Row-Column Implementation

Most of the parallel implementations of the Row-Column method have been realised by either using arrays of processing elements with a small memory (Siegel, 1981) or building a special purpose hardware (Lui, 1983). Other developers who used Transputers (Taylor, 1984, Harp *et al.*, 1985) commonly under-utilised on-chip memory by employing a number of Transputers to calculate a single 1D FFT. The implementation described here takes the full advantage of the Transputer on-chip memory, supplemented by an additional 1 Mb of off-chip memory, by carrying out several 1D FFTs on one Transputer. The resulting system achieves high computational efficiency and shows good performance.

A major bottle-neck in the Row-Column implementation of the 2D FFT is a matrix transposition phase which normally requires a secondary memory and therefore causes a traffic congestion between the main and the secondary memory. This problem is avoided here by using an efficient matrix transposition algorithm due to Eklundh (1972) which takes advantage of the large memory associated with each Transputer. The overall data traffic, inherent in this method, can be further minimised by dynamically reconfiguring the network with the use of the programmable link switch C004 (INMOS, 1989).

6.5.1 Row-Column method

The Row-Column method (2D FFT RC) is based on the 1D FFT (Cooley & Tukey, 1965) and a matrix transposition technique; its derivation can be explained in the following way.

If, as above, $N_1 = N_2 = N$, the equation (6.7) can be rewritten as

$$X(k_1,k_2) = \sum_{n_1=0}^{N-1} \left[\sum_{n_2=0}^{N-1} x(n_1,n_2)\, W_N^{n_2k_2}\right] W_N^{n_1k_1} \qquad (6.8)$$

As it can be seen from this equation, the summation in brackets is 1-dimensional Fourier transform along the n_2 dimension:

$$G(n_1,k_2) = \sum_{n_2=0}^{N-1} x(n_1,n_2)\, W_N^{n_2k_2} \qquad (6.9)$$

By substituting the expression in square brackets in equation (6.8) by the left side of equation (6.9), the 2-dimensional Fourier transform can be expressed as

$$X(k_1,k_2) = \sum_{n_1=0}^{N-1} G(n_1,k_2)\, W_N^{n_1k_1} \qquad (6.10)$$

Equation (6.9) means that each row of G is the 1-dimensional Fourier transform of the corresponding row of x, and each column of X is the 1-dimensional Fourier transform of the corresponding column of G. It follows from equation (6.10) that the 2-dimensional transform can be derived by applying 1D FFT along the columns of G. Thus the implementation of this method relies on performing 1D FFT on each row of x, by using any of the 'in-place' methods (Brigham, 1974), and then applying the same procedure along each column of G. If matrix G is transposed, i.e. each row in G becomes a column in the new transposed G^T, then identical 1D FFT can be applied to the rows of G^T.

The number of complex multiplications needed to perform 1D FFT is $\frac{N}{2}\log N$ (Brigham, 1974), therefore the number of complex multiplications needed for the 2D FFT using the Row-Column method can be evaluated by multiplying the above figure of 1D FFT with 2N, giving $N^2 \log N$.

6.5.2 Parallel implementation of the Row-Column method

The Row-Column method is implemented here using a network of Transputers (Mansoor & Claridge, 1989a, 1989b). The input data is supplied as a 2-dimensional matrix of real values. The host Transputer fetches the data matrix and sends sub-matrices to the network. The sub-matrix consists of a number of rows of the original matrix. The number of rows (R) being sent to a single Transputer depends on the number of Transputers (P) in the network and on the matrix size $2^N \times 2^N$ and can be expressed by the relation $R = \frac{N}{P}$.

The Transputers are interconnected as a mesh; however in the data transfer stage the mesh is used in a pipeline fashion, as shown in figure 6.5.

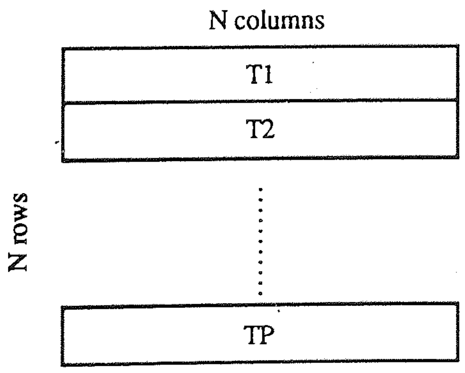


Figure 6.5 The distribution of subimages among the Transputers.

Each Transputer applies 1D FFT along the rows of its sub-matrix and on the completion of this task it contains the coefficients of the 1D FFT for all its rows. These coefficients form a 2-dimensional matrix which now has to be transposed so that the second 1-dimensional FFT can be applied in exactly the same way as before, that is along the rows of the transposed matrix (which are the columns of the matrix before transposition).

Matrix transposition is the only step which does involve data exchange among the Transputers in the network. The method described by Eklundh (1972) is an efficient and fast method, well suited for parallel implementation. It is based on a 'divide and conquer' strategy, where increasingly smaller portions of a 2-dimensional matrix are interchanged, as explained below.

Let X be the 2-dimensional matrix to be transposed, and let it consist of $2^N \times 2^N$ points. This matrix can be partitioned into four sub-matrices

$$X = \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix}$$

where X_{ij} is a sub-matrix of $2^{N-1} \times 2^{N-1}$ points.

Assuming that the origin of data is the upper left point of the matrix, the resulting matrix X^T (the transpose of X) takes the form

$$X^T = \begin{bmatrix} X_{00}^T & X_{10}^T \\ X_{01}^T & X_{11}^T \end{bmatrix}$$

where ^{T} represents the transposition.

The same procedure is subsequently applied to each of the four sub-matrices until a partition consisting of a single element is reached. As the upper right and the lower left partitions are interchanged in each step of the method, after $\log_2 N$ steps the entire matrix is transposed. An example illustrating this method is shown in figure 6.6 for 4x4 matrix.

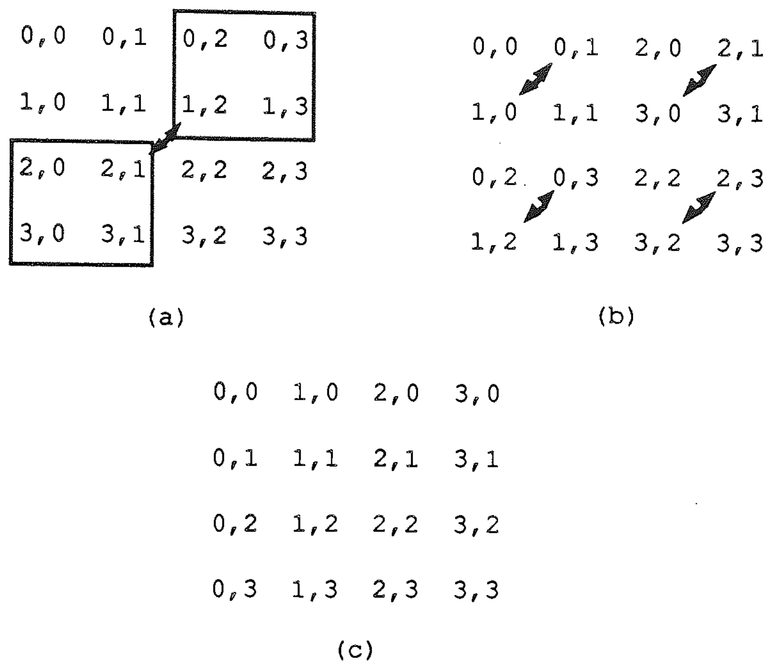


Figure 6.6 Matrix transposition procedure for 4x4 image.

The 1D FFT is applied again to the rows of the transposed matrix. As before, each Transputer carries out this operation on its local data and calculates the required coefficients of the 2-dimensional Fourier transform.

6.5.3 Matrix transposition protocol

The implementation of the matrix transposition protocol has to consider two cases: one, where data has to be transferred between two different Transputers; and the second, where data transfer takes place within a single Transputer.

The number of the algorithm steps (S) in the first case depends on the number of Transputers (P) and equals $S = \log_2 P$. (NB. if the number of rows of data $N<P$,

up to N Transputers can be used). In each step each Transputer communicates with one of its neighbours and data exchange takes place. Table 6.2 shows an example of the communications taking place for a mesh of 16 Transputers. In this example there are four steps, and the scheme of intercommunication is shown for each step.

step 0	step 1	step 2	step 3
T0-T8	T0-T4	T0-T2	T0-T1
T1-T9	T1-T5	T1-T3	T2-T3
T2-T10	T2-T6	T4-T6	T4-T5
T3-T11	T3-T7	T5-T7	T6-T7
T4-T12	T8-T12	T8-T10	T8-T9
T5-T13	T9-T13	T9-T11	T10-T11
T6-T14	T10-T14	T12-T14	T12-T13
T7-T15	T11-T15	T13-T15	T14-T15

Table 6.2 The intercommunication between the Transputers in matrix transposition algorithm.

The number of data packets to be transferred between any two Transputers in each direction depends on the step number. If the number of data packets is C(s), where s is the step number, then C(s) = 2s for $0 \leq s \leq S-1$.

The final steps of matrix transposition require data interchanges only within a Transputer.

6.5.4 Network description

The block diagram of the network is the same as in figure 5.9 and shown again in figure 6.7.

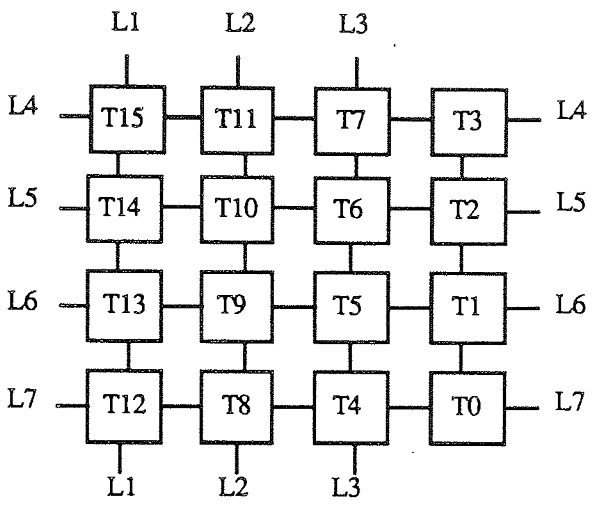


Figure 6.7 Network of Transputers for the 2DFFT RC.

The host Transputer acts as a controller of the network. It fetches the data from a filing system and distributes it across the network. The program stored in each Transputer contains five processes running in parallel: the 'data processing' process and the four 'router' processes, as shown in figure 6.8 below; full details of this model were described in section 5.5. Each Transputer in the network has a unique label. For data communication, each packet contains a tag which identifies a destination Transputer and the type of the data in the packet.

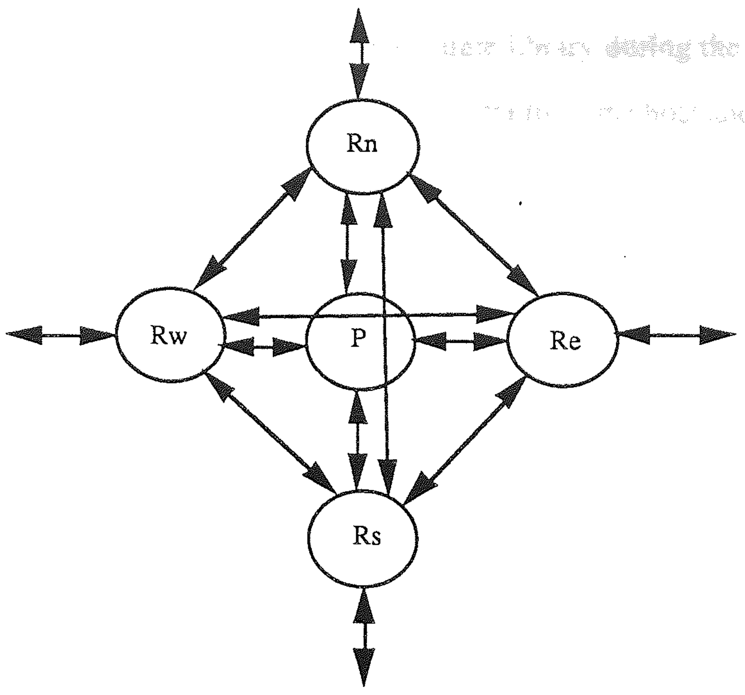


Figure 6.8 Occam model for a Transputer in the array.

The five processes running in parallel can be implemented using PRI PAR construct of OCCAM:

```
PRI PAR
  PAR
     Rw
     Re
     Rn
     Rs
  P
```

The Process procedure contains the whole data processing operation assigned to the Transputer while the routers procedures are responsible for moving the data between Transputers. The PRI PAR structure is used, since for the routers high priority is needed to ensure speedy data distribution. The router processes will not consume much time of the processor since for data input or output they need very little time to initialise the DMA circuit to complete the job.

The data packets are defined and put in a separate library during the development time. For example, the data packets for moving data from the host and between the Transputers are

data.type; P; n :: x

where data.type shows the data type of this data packet, P is the number of a target Transputer, n is the size of the array x being transmitted. In each process there is ALT construct to wait for any message coming to it. Within each channel waiting for communication, a CASE feature is used to identify which type of data packet has arrived.

The process P is responsible for mainly two actions:

1) handling the routers.

2) handling each case within the ALT construct according to each recieved protocol type.

Each process P has four channels to connect it to the four routers, so the main procedure construct is as follows;

```
WHILE continue
 ALT
 ch1 ? CASE
      data.packet.type.1
            (processing)
      data.packet.type.2
            (processing)
 ch2 ? CASE
      (same as ch1)
 ch3 ? CASE
      (same as ch1)
 ch4 ? CASE
      (same as ch1)
```

The WHILE construct has been used to let the processing continue as long as a logical value 'continue' logic value is TRUE; its value can be changed by the host for processing termination.

The construct of the router is similar except that there is no processing but just routing the data to its right targets. The various types of data packets are as follows;

w.data; P; n :: data; nrow; ncol

where

> w.data is the data packet type
>
> P is the number of the target Transputer
>
> n is the size of the data to be sent
>
> nrow is the number of rows of the data
>
> ncol is the number of column of the data

w.acknowledge; P; data.received

where

> w.acknowledge is the data type for data acknowledgement to be sent to the host so there is no need to put the target Transputer number.
>
> P is the number of the sender
>
> data.received is the logic value to acknowledge when the data is successfully received (TRUE) or not (FALSE).

w.terminate

This is to change the value of continue from TRUE to FALSE to terminate the processing.

stage.number; stage

This is to send the stage number within the matrix transposition protocol.

data.transposition.sender; P; n:: x and

data.transposition.receiver; P; n:: x

As it is shown in Table 6.2 above, in each step two columns of Transputers need to communicate; the first column is called 'senders' and the second is called 'receivers'. The above two packets are for the senders and receivers respectively. In practice both data packets are included in the routines of T1 to T15 since in some stages they are senders while in others they are receivers.

6.5.5 Performance estimation

The main factor in the assessment of the performance of an implementation is the speed-up of processing: the greater the speed-up, the better performance. The speed-up can be estimated as follows:

If t1 is the time of calculating 2D FFT on a single Transputer, then the time of this calculation on a network of Transputers P, excluding the matrix transposition, is equal to t1/P.

The time tc spent on matrix transposition on P Transputers can be estimated as

$$tc = \frac{d}{v},$$

where d is the amount of data to be interchanged between Transputers:

$$d = \left(\frac{N^2}{P}\right)\log_2 P$$

and v is the speed of data transfer, which is the link speed of the Transputer.

Thus the total time of processing of the 2D FFT on P Transputers is

$$t = \frac{t1}{P} + tc.$$

The communication overhead (Coh) can be defined as the ratio of tc to t1/p;

$$Coh = \frac{tc}{(\frac{t1}{P})}$$

The speed-up Q can be expressed as

$$Q = \frac{t1}{t} = \frac{P}{1+Coh}.$$

This means that the speed-up is directly proportional to P and inversely proportional to Coh. Figure 6.9 shows the graphical relationship between speed-up and Coh.

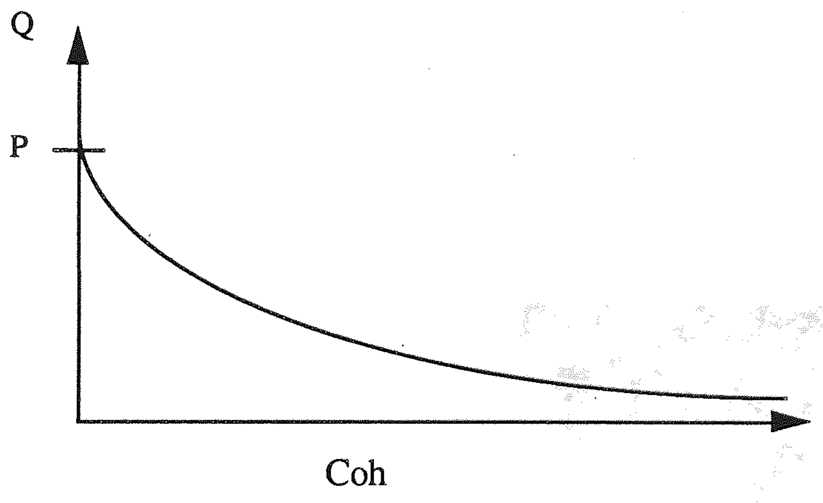
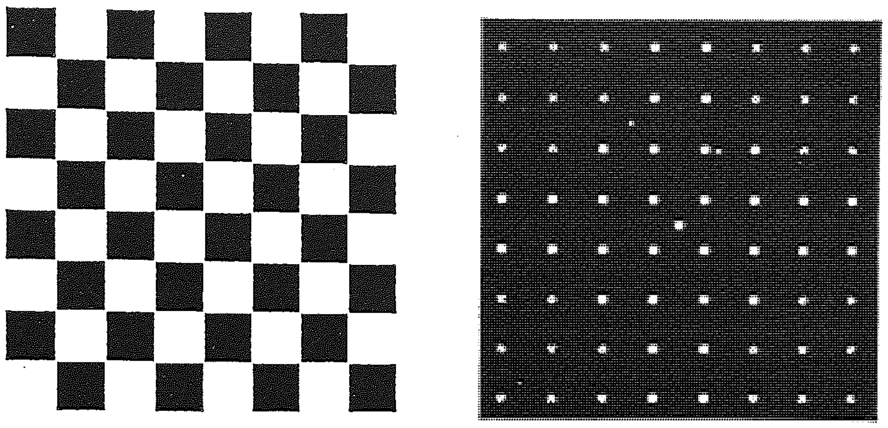


Figure 6.9 The graph of the speed up with respect to the communication overhead.

6.5.6 Results and analysis

The 2DFFT RC algorithm has been implemented on a network of Transputers and the initial estimates were tested for four image sizes and two different mesh sizes. The timer of the host is used to measure the total time of processing which is from the instant when the host starts to distribute the subimages to the instant when the

the last Transputer has just
last byte of the last subimage result is received. Figure 6.10 shows two images and their Fast Fourier Transform.


(a) Checker board (b) 2DFFT of (a)

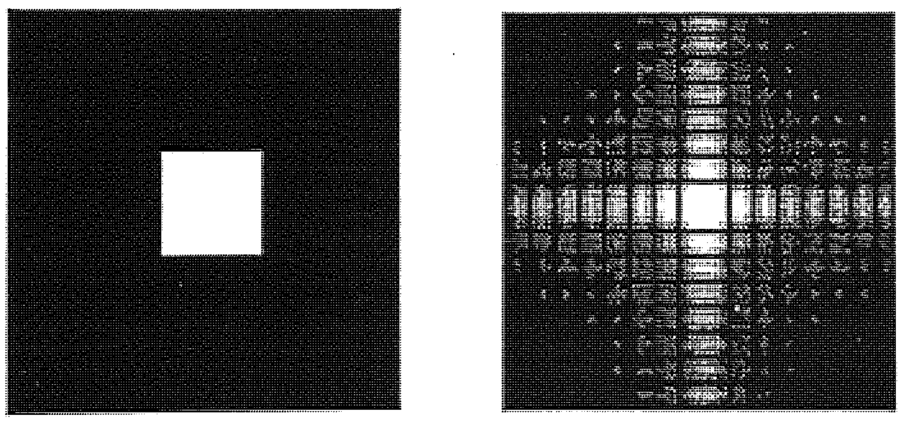

(c) Square box (d) 2DFFT of (c)

Figure 6.10 The Fast Fourier Transform of two images.

For the measurement of the communication overhead (Coh), the timer of Transputer numbered zero in the network shown in figure 6.7 on page 113 is used. This does not give the exact communication overhead but an approximation of it. For actual measurement of communication overhead, it should be the time from the instant when the first Transputer in the network is ready to apply the intercommunication protocol to the instant where the last Transputer has just finished the intercommunication protocol. This can be achieved by recording the timers of the Transputers in the network and then making comparison between them to find out the actual communication overhead. However there is no big difference between the two because all the Transputers implement similar operations apart from some differences due to the longer path for receiving data and this will not affect the result.

Table 6.3 shows the total time of processing Tt for the 2DFFT RC implementation, the actual time of processing of the two 1DFFT algorithms, Tp1 and Tp2, and the time of processing spent on the matrix transposition Tc which represents the intercommunication process. The graph for total time of processing for three arrays of Transputers is shown in figure 6.11 which follows the table. The graph of the communication overhead is shown in figure 6.12.

image side	Tt	Tp1	Tp2	Tc
8	0.065	0.029	0.029	0.002
16	0.367	0.174	0.174	0.006
32	1.924	0.93	0.924	0.021
64	9.538	4.645	4.617	0.091
128	45.558	22.281	22.153	0.406
256	217.6	100.26	100.26	1.827

(a) Time measurements for a single Transputer

image side	Tt	Tp1	Tp2	Tc
8	0.022	0.007	0.007	0.004
16	0.105	0.043	0.043	0.01
32	0.519	0.232	0.231	0.03
64	2.515	1.161	1.154	0.111
128	11.88	5.571	5.538	0.457
256	54.907	25.984	25.836	1.895

(b) Time measurements for four Transputers

image side	Tt	Tp1	Tp2	Tc
16	0.043	0.011	0.01	0.012
32	0.167	0.058	0.057	0.027
64	0.741	0.291	0.288	0.085
128	3.383	1.394	1.384	0.301
256	15.264	6.499	6.457	1.144

(c) Time measurements for sixteen Transputers

Table 6.3 Time measurements for 2DFFT RC implementation on array of single, four, and sixteen Transputers.

Time is given in seconds.

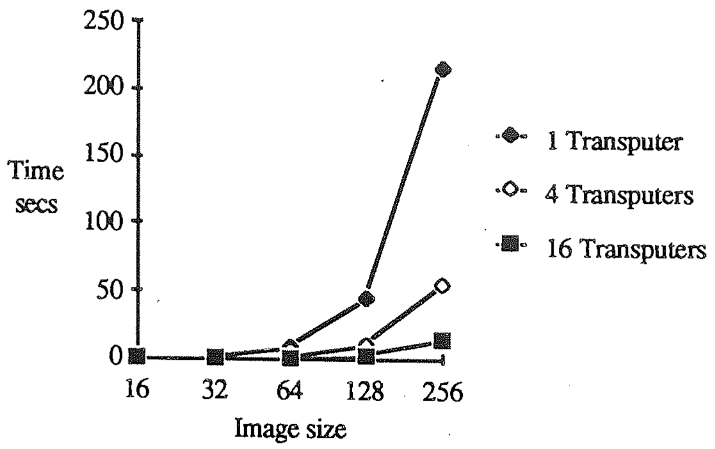


Figure 6.11 Time for 2DFFT RC on a Transputer array.

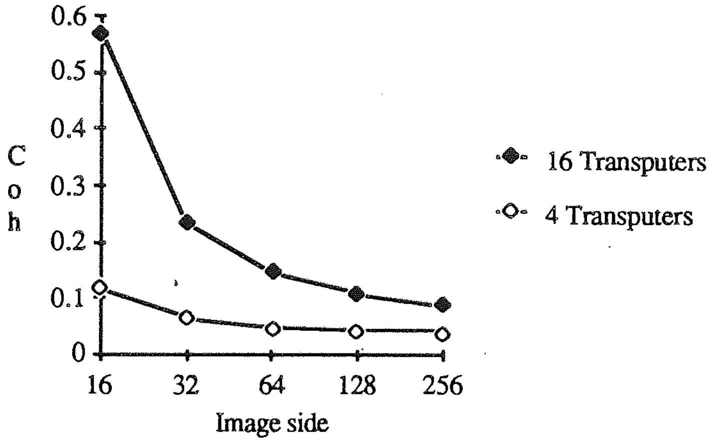
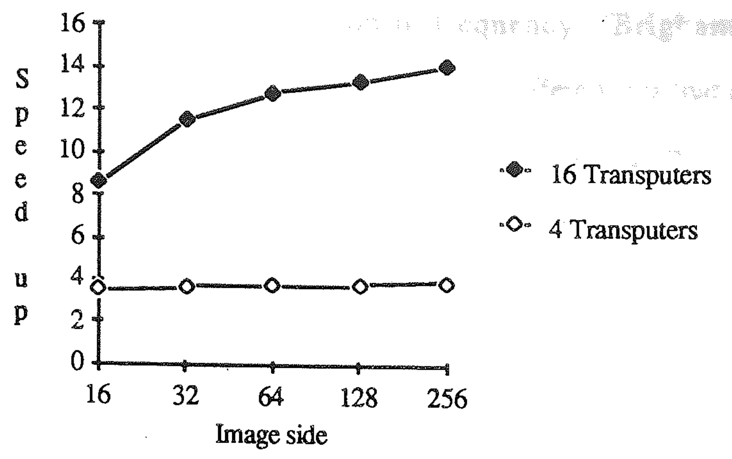
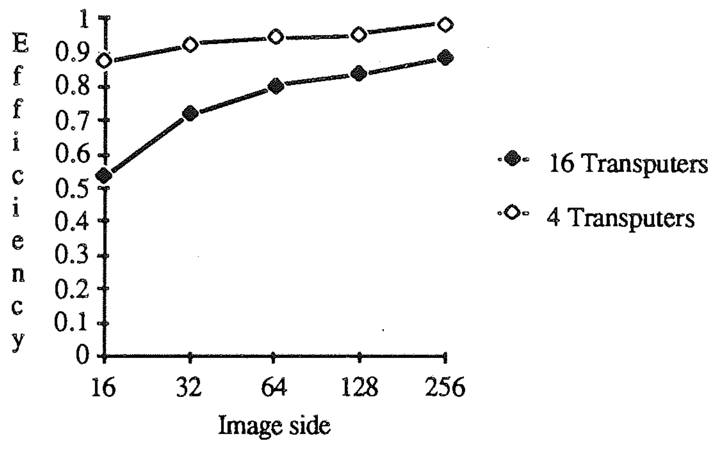


Figure 6.12 Communication overhead in the 2DFFT RC implementation.

Figure 6.13 (a) below shows that the speed-up increases with the increase of the data size, and this is because the communication overhead Coh decreases with the increasing data size. This agrees with the theoretical estimation of figure 6.9. For a given data size the speed-up increases with the number of processors used. Figure 6.13 (b) shows that efficiency increases with the increase of the data size, but for a given data size it decreases with the increase of the number of processors. This is because the increase in the number of processors increases the communication overhead Coh in the larger network.



(a)



(b)

Figure 6.13 Performance of 2DFFT RC implementation.

6.6 Vector-Radix Implementation

The Vector-Radix algorithm (2D FFT-VR), which was chosen for this implementation, is based on the divide-and-conquer strategy described by Dudgeon & Mersereau (1984). A complete 2D FFT is broken down into successively smaller 2-dimensional transforms until a trivial 2-dimensional FFT is reached. The basic operation in this algorithm is the so-called 'butterfly' operation which requires four inputs and produces four outputs. There are two versions of the algorithm:

decimation-in-time and decimation-in-frequency (Brigham, 1974). The computational effort is the same for both; the main difference is that for the first one there is no multiplication needed in the first stage of the algorithm, while for the second one there is no multiplication needed in the last stage. This difference is important when implementing this algorithm on a pyramid architecture as will be shown in the following section.

Divide and conquer strategy is used for 1-dimensional FFT computation and it is applied to the 2-dimensional FFT algorithm. A 2-dimensional FFT is broken down into successively smaller 2D blocks until a trivial 2D FFT block is reached which is very easy to evaluate.

The decimation-in-time version of the algorithm can be derived by expressing an (NxN)-point FFT in terms of (N/2xN/2)-point FFT. For direct calculation of 2D FFT.

$$X(k_1, k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2)\, W_N^{(n_1k_1+n_2k_2)} \qquad (6.11)$$

for $0 \le k_1 \le N\text{-}1, 0 \le k_2 \le N\text{-}1$

where $x(n_1, n_2)$ is the data in the spatial domain, $X(k_1, k_2)$ is the data in the frequency domain, and

$$W_N = e^{(-j\frac{2\pi}{N})}$$

The FFT summation of equation 6.11 can be decomposed into four summations: one over those samples of x for which both n_1, n_2 are even; one for which n_1 is even and n_2 is odd; one for which n_1 is odd and n_2 is even; and one for which both n1 and n2 are odd. This gives

$$X(k_1, k_2) = S_{00}(k_1, k_2) + S_{01}(k_1, k_2)W_N^{k_2} + S_{10}(k_1, k_2)W_N^{k_1} + S_{11}(k_1, k_2)W_N^{(k_1+k_2)} \qquad (6.12)$$

where S_{00}, S_{01}, S_{10}, and S_{11} are periodic in (k_1, k_2) with horizontal and vertical periods N/2,

$$\begin{aligned}
S_{00}(k_1, k_2) &= \sum_{m_1=0}^{N/2} \sum_{m_2=0}^{N/2} x(2m_1, 2m_2)\, W_N^{(2m_1k_1+2m_2k_2)} \\
S_{01}(k_1, k_2) &= \sum_{m_1=0}^{N/2} \sum_{m_2=0}^{N/2} x(2m_1, 2m_2+1)\, W_N^{(2m_1k_1+2m_2k_2)} \\
S_{10}(k_1, k_2) &= \sum_{m_1=0}^{N/2} \sum_{m_2=0}^{N/2} x(2m_1+1, 2m_2)\, W_N^{(2m_1k_1+2m_2k_2)} \\
S_{11}(k_1, k_2) &= \sum_{m_1=0}^{N/2} \sum_{m_2=0}^{N/2} x(2m_1+1, 2m_2+1)\, W_N^{(2m_1k_1+2m_2k_2)}
\end{aligned} \qquad (6.13)$$

Using the fact that

$$W_N^{N/2} = -1,$$

the following identities can be derived from equation 6.12 and a set of equations 6.13:

$$\begin{aligned}
X(k_1, k_2) &= S_{00}(k_1, k_2) + S_{01}(k_1, k_2)W_N^{k_2} + S_{10}(k_1, k_2)W_N^{k_1} \\
&\quad + S_{11}(k_1, k_2)W_N^{(k_1+k_2)} \\
X(k_1+N/2, k_2) &= S_{00}(k_1, k_2) + S_{01}(k_1, k_2)W_N^{k_2} - S_{10}(k_1, k_2)W_N^{k_1} \\
&\quad - S_{11}(k_1, k_2)W_N^{(k_1+k_2)} \\
X(k_1, k_2+N/2) &= S_{00}(k_1, k_2) - S_{01}(k_1, k_2)W_N^{k_2} + S_{10}(k_1, k_2)W_N^{k_1} \\
&\quad - S_{11}(k_1, k_2)W_N^{(k_1+k_2)} \\
X(k_1+N/2, k_2+N/2) &= S_{00}(k_1, k_2) - S_{01}(k_1, k_2)W_N^{k_2} - S_{10}(k_1, k_2)W_N^{k_1} \\
&\quad + S_{11}(k_1, k_2)W_N^{(k_1+k_2)}
\end{aligned} \qquad (6.14)$$

It can be seen that S_{00}, S_{01}, S_{10}, and S_{11} can be obtained by evaluating a (N/2 x N/2)-point FFT. The computation represented by equations 6.14 is called a radix-(2x2) butterfly as shown in figure 6.14.

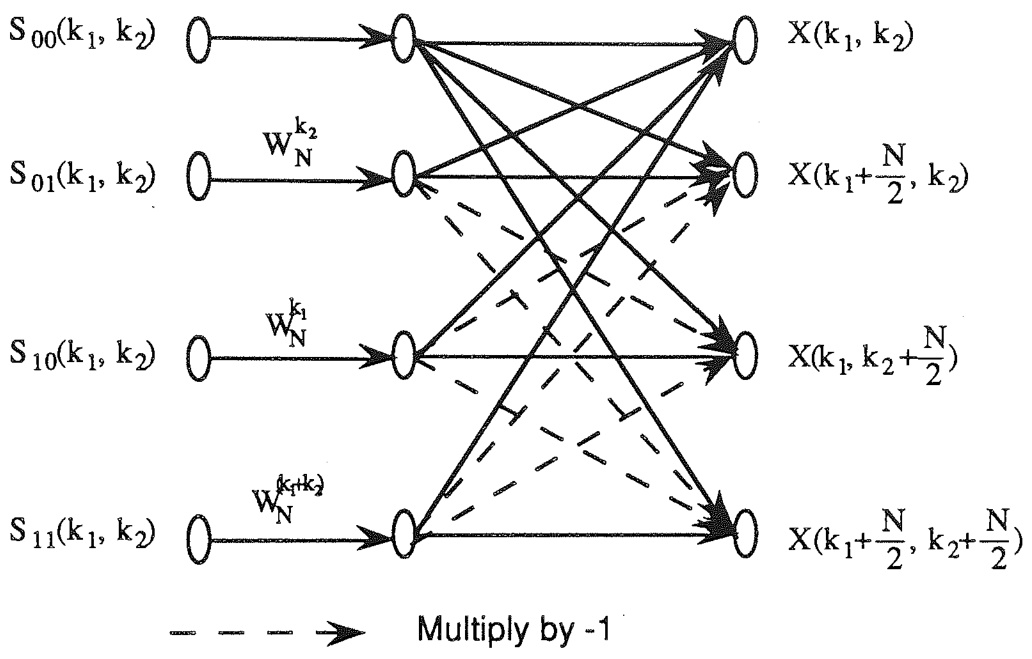


Figure 6.14 Butterfly operation.

It is clear from figure 6.14 that each butterfly requires three complex multiplications and eight complex additions and to compute all the samples of X from S_{00}, S_{01}, S_{10}, and S_{11} requires the calculation of $(N^2/4)$ butterflies. The number of steps to complete the computation is equal to (log N). Thus the number of multiplications needed to complete the computation is

$$C = \frac{3N^2}{4} \log N$$

which is 25% fewer than that needed for the Row-Column decomposition.

6.6.1 Pyramid architecture for the Vector-Radix Method

The Vector-Radix method can be effectively implemented using the pyramid data structure (Mansoor & Claridge, 1989c) (see section 3.3.2 for a general discussion of a pyramid architecture). The first stage takes the data in so-called 'bit-reversed' order (Dudgeon & Mersereau, 1984); the data is divided into subimages of size $N/2^{(S-i)}$, where N is the image size in one dimension (it is assumed that N is a power of 2), S is the number of the last stage, and i is the stage number, where $0 \leq i \leq S$. The size of each subimage in the first stage is equal to two, so there is one butterfly operation per subimage. In the successive stages, each four subimages-siblings merge to create a larger one which is called the parent. The number of butterfly operations per each subimage is equal to four times that of the previous stage; the total number of butterfly operations in each stage is $N^2/4$. Figure 6.15 illustrates this procedure for the image of size 8x8.

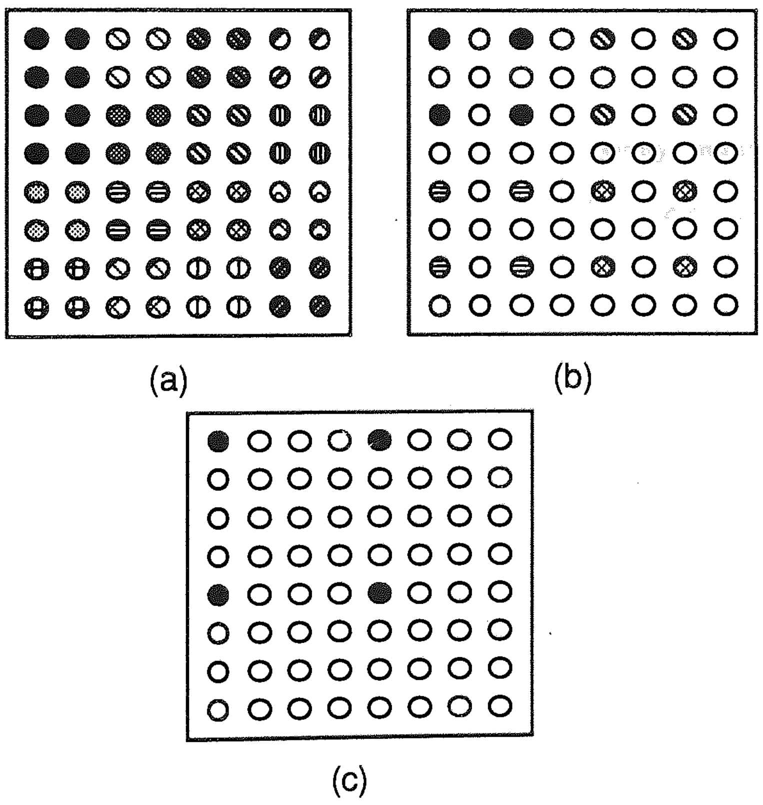


Figure 6.15 Three-level pyramid implementation: (a) stage 0, (b) stage 1, and (c) stage 2.

Figure 6.15 (a) indicates the sets of inputs for each butterfly operation by different patterns. Each set of four inputs lies within a different subimage, and because the subimage size is only 2x2, only one butterfly operation is applied. Figure 6.15 (b) shows stage 1. Here the subimage size is 4x4 and so four butterfly operations are required. The inputs for the first butterfly operation are shown with shading. The second butterfly operation uses the four right neighbours of these pixels as input, the third operation uses the four lower neighbours, and the fourth uses the four diagonal right neighbours. Figure 6.15 (c) shows stage 2 where there is only one subimage, which is the size of the entire image. The first four butterfly inputs are shown; the successive inputs to the butterfly will be successively displaced from these positions, as in stage 1, giving the sixteen input data sets required.

For implementing a Vector-Radix method a parallel algorithm has been developed using both a decimation-in-time and a decimation-in-frequency strategy and a pyramid architecture. For the decimation-in-frequency strategy, the image data is first bit-reversed. The processing starts at the bottom level of the pyramid (level 0) where each processor has access to a sub-image of size equal to N^2/P^2, where N^2 and P^2 are image and pyramid sizes respectively. Each processor executes a number of stages of the Vector-Radix algorithm, up to the size of the sub-image in it, and then sends the result to the next level of the pyramid. In each successive level only one stage of the algorithm is executed. The algorithm is very efficient as no communication is necessary between processors on a single level and no multiplication is needed on the highest level of the pyramid. For the decimation-in-time strategy, the data is first sent to the root because there is no multiplication needed in the first stage of the algorithm, so a top-down scheme is used. The following example illustrates the implementation method for a 2x2 matrix. The diagram of the butterfly operation for this matrix is shown in figure 6.16 on the next page.

A pyramid of 2 levels (i.e of size 2) is used. First, the data is arranged in the bit-reversed order and sent to the base of the pyramid. Each Transputer applies one butterfly operation and sends the results to the higher level, in this case the root. The root Transputer implements the four butterfly operations, but since there is no multiplication needed in the root a bottleneck is not created. For larger matrix sizes each butterfly in the diagram would indicate an appropriate number of butterfly operations.

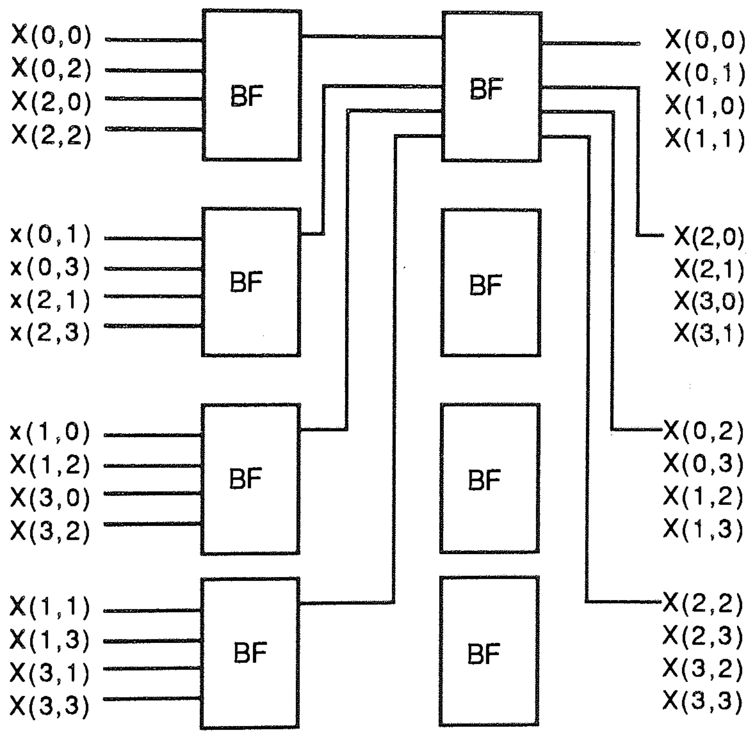


Figure 6.16 Butterfly diagram for Vector-Radix method: BF indicates a butterfly operation.

6.6.2 Theoretical Estimation of Performance

A theoretical estimate has been made to evaluate the performance of the parallel algorithm on a pyramid architecture, and in particular to find out how the performance is affected by the pyramid size and the image size within that pyramid.

Each processor in level zero (base level) applies a Vector-Radix algorithm to its own sub-image. In this computation the number of stages is equal to log(N/P) and the number of complex multiplications is equal to

$$C_0 = 3/4\ (N^2/P^2)\ \log(N/P)$$

On level one, only one stage of the algorithm will be executed, so the number of complex multiplications is equal to

$$C_1 = 3/4\ (2N/P)^2$$

The results are then passed to the next level until the whole image reaches the root processor. In the root processor no multiplication is required. The only drawback of this implementation is that the time needed to compute one stage in level i (where i is neither the base level nor the root level) is approximately four times less than the time of processing of the stage (i+1); for example for i=2

$$T_2 = 3/4\ (2N/P)^2\ , \qquad T_3 = 3/4\ (4N/P)^2$$

hence $T_2 / T_3 = 1/4$

where T_2 and T_3 are time of computation for the second and the third levels respectively: please note that since time of multiplication affects most the overall computing time, C_i and T_i are used interchangeably.

As an example of the theoretical estimation a pyramid of size 16 (P = 4) will be considered. The time estimate takes the form

$$T_0 = 3/4\ (N/4)^2 \log(N/4) \ \text{ and } \ T_1 = 3/4\ (N/2)^2$$

T_2 is very small because this level is the root level which implements the last stage of the method where no multiplication takes place. For this particular pyramid, the best performance is achieved when the ratio T_0 / T_1 is equal to one:

$$T_0 / T_1 = \log(N/4) / 4 = 1$$

thus giving N = 4.Figure 6.17 is the graph of this equation showing the relationship between T_0 / T_1 and the image size.

The analysis has shown that a pyramid of size 4x4 (3 levels) gives the best performance for this particular parallel implementation of the 2D FFT algorithm. This is due to the drawback mentioned above for pyramid of sizes more than 3 levels. Subsequently, the relationship between the ratio of the processing time on level 0 and on level 1 and the image size has been examined. From this analysis it has been noted that the best performance is achieved for an image of size 64x64. However, the performance for images of size up to 1024x1024 is still good and does not degrade to an unacceptable level.

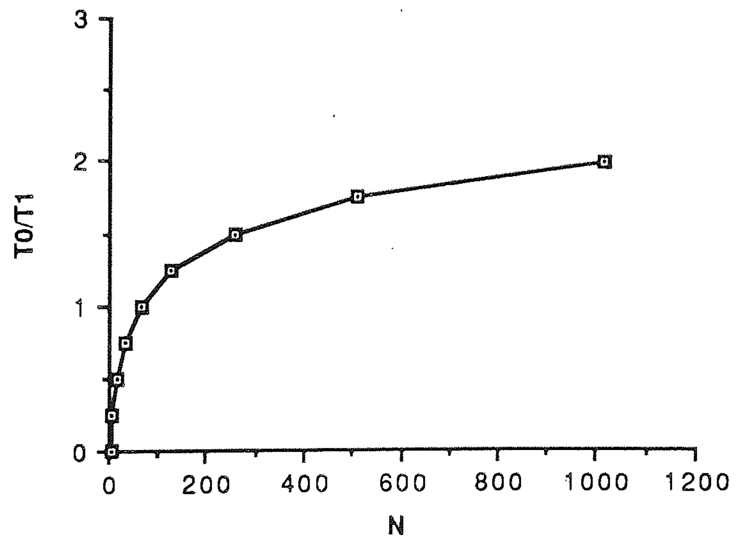
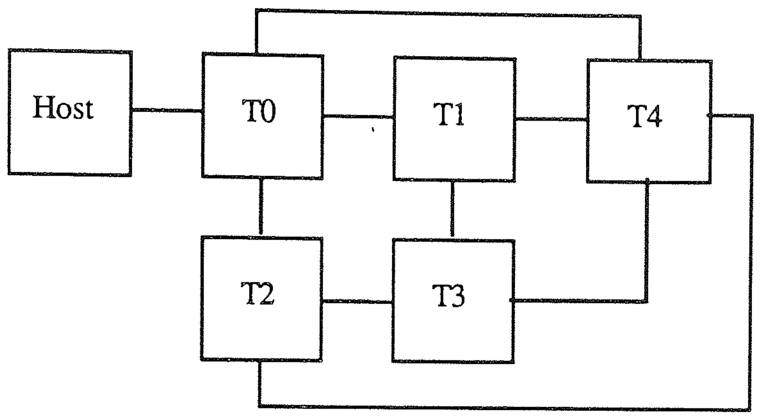


Figure 6.17 Estimation of T_0/T_1 versus image size for pyramid of size 16.

6.6.3 Parallel Implementation and Performance Evaluation

A pyramid of size four of Transputers is used for the implementation of the 2DFFT-VR algorithm. Figure 6.18 shows the block diagram of the implementation.



(a)

Figure 6.18 Pyramid of size four.

T0, T1, T2, and T3 represent the base level of the pyramid (level 0). The subimages are distributed to T0-T3 by the host Transputer according to figure 6.18 (b). T4 is the root Transputer of the pyramid (level 1).

In level zero, each Transputer processes the subimage, which has been sent by the host Transputer, and sends the result to the root Transputer. On the completion of the processing the Transputers T0-T3 send a message to T0. T0 sends a request to the host Transputer for another image after receiving the messages from all Transputers in level zero. The timer of T0 is used to measure the time spent by the Transputers of level zero on processing the subimages of an image.

T4 receives the subimage results from the Transputers of level zero and applies the last stage of the algorithm as described in section 6.6.1. The timer of T4 is used to measure the time of processing of each complete image. It measures the time of processing at the root level as well.

The timing of the 2D FFT implementation has been performed on a sequence of four similar images. For the purpose of timing real type data only has been used since for complex data the time would be double. Table 6.4 shows the time of processing for the four images. In Tables 6.4(a), T01 to T04 are the time of processing on level zero for images one to four respectively, Tt1 to Tt4 are the total time of processing for images one to four respectively. In table 6.4(b) T0 and T1 are the time of processing on levels zero and one respectively.

image side	T01	T02	T03	T04	Tt1	Tt2	Tt3	Tt4
16	0.059	0.06	0.059	0.06	0.074	0.059	0.06	0.059
32	0.351	0.351	0.351	0.351	0.41	0.351	0.351	0.348
64	1.867	1.867	1.868	1.867	2.098	1.868	1.867	1.867
128	9.375	9.375	9.375	9.375	10.288	9.375	9.375	9.375

(a)

image side	T0	T1	T1/T0
16	0.036	0.006	0.166
32	0.225	0.032	0.142
64	1.184	0.124	0.104
128	5.91	0.48	0.081

(b)

Table 6.4 Time of processing 2DFFT VR on pyramid of size four.

Time is given in secs.

Figure 6.19 shows the total time of processing the 2DFFT VR algorithm on a single Transputer and Transputer pyramid of size four.

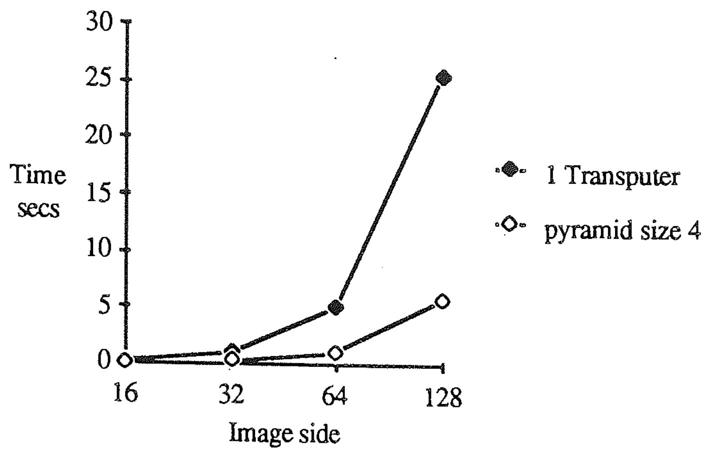


Figure 6.19 Time of processing for 2DFFT VR algorithm.

Several performance graphs have been plotted to evaluate the implementation performance. Figure 6.20 shows the relationship between the ratio of time of processing of level zero and level one with respect to the image sizes. This graph is important in the pipeline architecture to evaluate the load balance in the system.

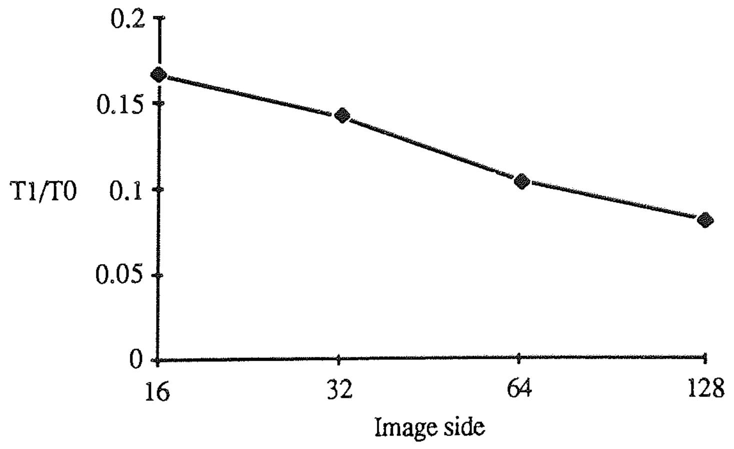
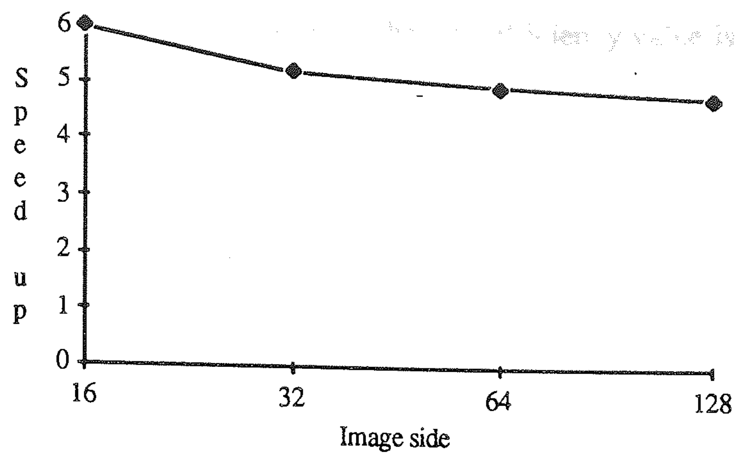
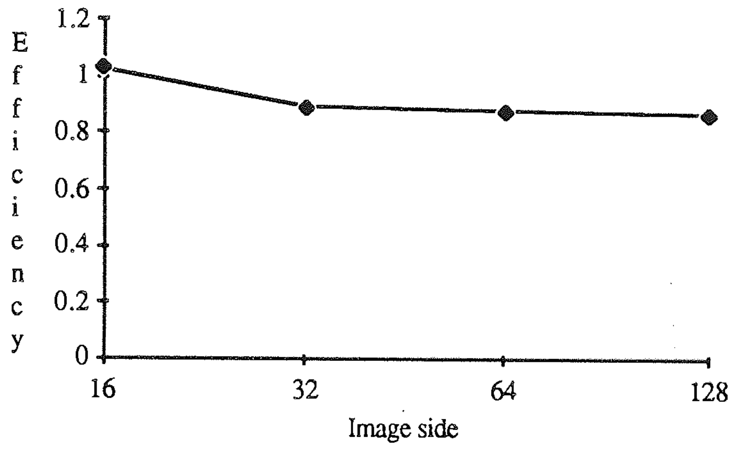


Figure 6.20 2DFFT VR: load balance of pyramid of size four.

The speed up and the efficiency are calculated using tables 6.4 (a)-(b) above. Figure 6.21 illustrates them.



(a)



(b)

Figure 6.21 Speed up and efficiency for the pyramid of size four.

Since there were not enough Transputers available in the department for building a pyramid of size sixteen, the performance of that pyramid was estimated, based on the actual processing of the algorithm but neglecting the time for data distribution. The performance of the implementation of the algorithm on a pyramid of size four was also estimated. This is just to compare the estimated performance with the actual performance shown in figure 6.20 and to justify the estimated performance of a pyramid of size sixteen. The only drawback of that estimation is that since the time of processing on a single Transputer (t_1) includes the time for data distribution and collection which shows t_1 time higher than that if the only actual processing is taken, and since efficiency = t_1/nt_n, the efficiency value in this estimation is slightly more than its real values and that why when the efficiency value is very close to one, it goes more than one.

The method of estimation is based on the main feature of the parallel algorithm proposed for the 2DFFT VR method. This feature is that there is no intercommunication between the Transputers of the same level. Accordingly, the time of processing of each level is equal to the time of processing of the subimage size of that level on one Transputer. The time measurements of various size of images have been achieved for the implementation of level 0 up to the root level. These measurements are shown in table 6.5. In this table Ts is the time of processing the algorithm on a single Transputer, T0 is the time of processing of level zero, Tm is the time of processing of any level which is neither level zero or the root level, and Tr is the time of processing of the root level. For T0, Tm and Tr, the time of processing is for the corresponding subimage size.

image side	Ts	T0	Tm	Tr
4	0.003	0.004	0.003	0.0005
8	0.025	0.045	0.08	0.002
16	0.173	0.211	0.056	0.008
32	0.968	1.11	0.22	0.03
64	5.092	5.756	0.888	0.131
128	25.117	27.865	3.55	0.525

Table 6.5 2DFFT VR: The actual time of processing for level 0 up to the root level for several image sizes.

Time is given in secs.

The times of processing for pyramids of sizes four and sixteen are taken from table 6.5 and shown in tables 6.6 and 6.7 below. In these two tables T0 and T1 are the time of processing in levels zero and one respectively.

image side	T0	T1	T1/T0
8	0.004	0.002	0.5
16	0.045	0.008	0.177
32	0.211	0.03	0.142
64	1.11	0.131	0.118
128	5.756	0.525	0.091
256	27.865	2.1	0.075

Table 6.6 2DFFT VR: The estimated time of processing on pyramid of size four.

Time is given in secs.

image side	T0	T1	T2
16	0.004	0.08	0.008
32	0.045	0.056	0.03
64	0.211	0.22	0.131
128	1.11	0.888	0.525

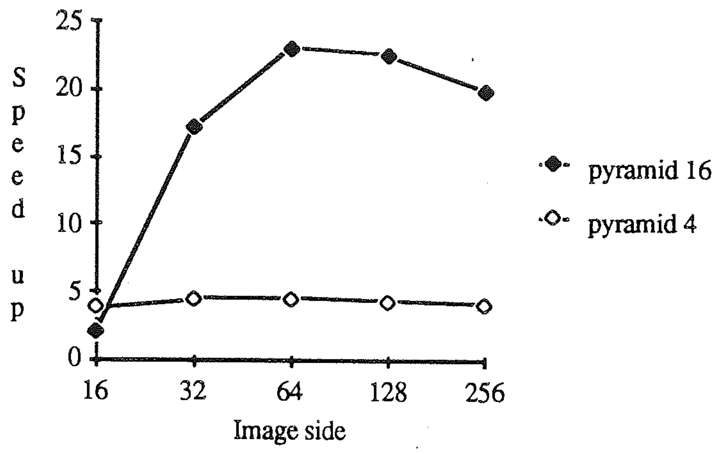
Table 6.7 2DFFT VR: Estimated time of processing on pyramid of size sixteen.

Time is given in secs.

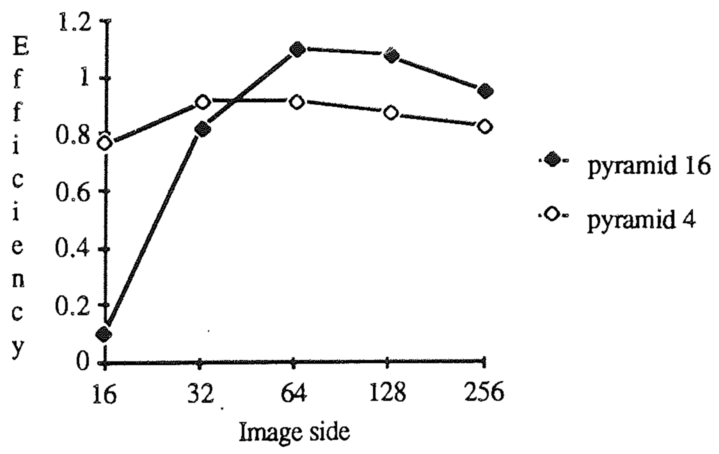
The estimates shown in these tables have been derived using the following assumptions. The time of processing of an image of size 128 on level 0 of a pyramid of size sixteen is the same as the time of processing of an image of size 32 on a single Transputer; the time of processing on level 1 is the same as the time of that on a single Transputer implementing one stage for image size 64; The time of processing on the root level is the same as the time of processing of the last stage of the image of size 128 on a single Transputer.

Table 6.6, with estimated results, is very close to table 6.4(b), which is based on the real time implementation, and this proves that the estimation procedure used here gives correct results.

The time of processing for the image is equal to the largest time among the times of processing of all the levels. This is because they are in pipeline. The performance of implementation of the two pyramids has been calculated and plotted in figure 6.22.



(a)

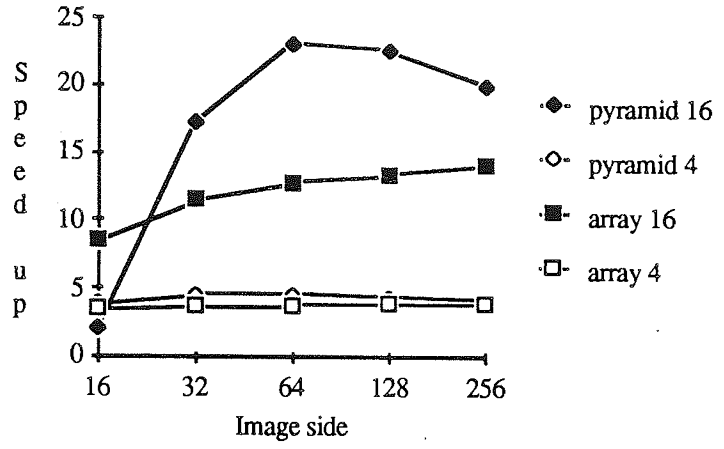


(b)

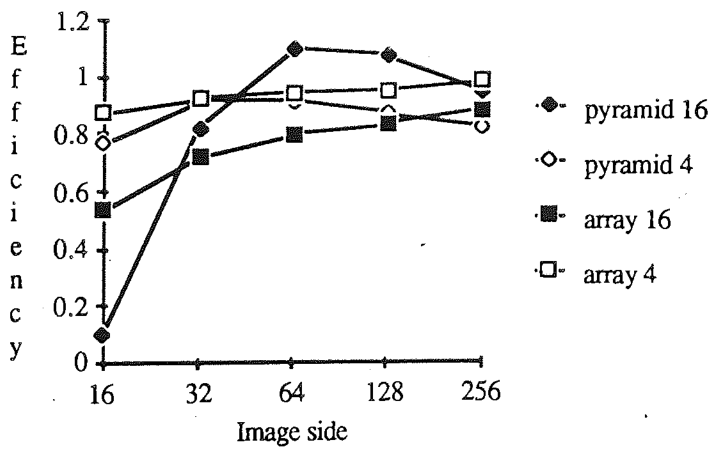
Figure 6.22 2DFFT VR: The speed up and the efficiency graphs for pipeline processing on pyramid of size four and sixteen.

These graphs show that the speed up achieved in the pyramid of size sixteen is superior. The maximum speed up is for image size of 64, which is similar to the theoretical estimation in figure 6.17.

For comparison purposes figure 6.23 shows graphs of speed up and efficiency with respect to image side for the array architecture (2DFFT RC method) and the pyramid architecture (2DFFT VR method), the graphs below combine the graphs in figure 6.13 and 6.22.



(a)



(b)

Figure 6.23 The speed up and efficiency graphs for the pyramid and the array architecture implementation of 2DFFT.

The above figure shows that speed up gained in the pyramid architecture is greater than that achieved by the array architecture and this is because of the more Transputers used in the pyramid architecture. The graph of efficiency shown in figure 6.23 (b) indicates that both architectures have approximately similar efficiency. Since the performance of both architectures is approximately similar then the decision of which one is better depends on the efficiency of the sequential algorithm when implemented on a single Transputer. Since the time of processing of the 2DFFT VR (table 6.5) on single Transputer is faster than that of 2DFFT RC (table 6.3 (a)), which agrees with the theoretical derivation in section 6.5.5, then the conclusion is that the 2DFFT VR method represents a better method for implementing the FFT on a network of Transputers.

6.7 Implementation of Convolution via FFT method

The implementation of the convolution using the FFT technique is achieved by applying the steps of the procedure outlined in the second part of section 6.1.2. Either of the methods of implementing the FFT described above can be used. Here the two methods are used for the convolution implementation and performance is evaluated and discussed.

6.7.1 Row-Column based Implementation

The array architecture used for the implementation of convolution is the same as that used for the implementation of the 2D FFT RC method since the convolution operation consists mainly of two Fast Fourier Transform operations. The 2D FFT of the kernel data is computed and divided into subkernels of size equal to that of the corresponding subimage size which is going to be sent during the operation of the 2D FFT. Each subkernel is stored in the corresponding Transputer. This is done to speed up the processing.

The system block diagram is the same as that of 2D FFT RC implementation which was shown in figure 6.7. The host fetches the input image from the filing system and divides it into several subimages and then distributes them between the Transputers in the network. The 2D FFT RC algorithm is applied as described in section 6.5.2.

The subimage result in each Transputer represents the Fourier transformed data of the original subimage. The next step of the convolution operation is that each transformed subimage is multiplied with the transformed subkernel of convolution which already resides in the Transputer.

The final step of the convolution operation is to apply the inverse 2DFFT operation on the resultant subimage data and send the final result back to the host Transputer.

The performance of the system has been measured for several image sizes. The timer of one of the Transputers in the network is used to measure the time spent on actual processing (the forward Fourier Transform and the inverse one) and that which is spent on the multiplication with the transformed subkernel in the frequency domain. Tables 6.8(a)-(c) on the next page show the time of processing on a single Transputer, on an array of four Transputers, and an array of sixteen Transputers respectively. Tt is the total time of processing, Tp is the time of processing the forward and the inverse FFT, and Tc is the time of multiplication with the transformed subkernel. It can be seen from the Tables that time Tc is very small with respect to Tp and can be neglected. Figure 6.24, which follows the tables, shows the graph of the total time of processing Tt for various image sizes and for the three Transputer arrays.

image side	Tt	Tp	Tc
16	0.726	0.699	0.012
32	3.817	3.741	0.042
64	18.959	18.63	0.182
128	90.660	89.324	0.812
256	421.597	401.12	3.654

(a) a single Transputer.

image side	Tt	Tp	Tc
16	0.2	0.172	0.02
32	1.009	0.931	0.06
64	4.930	4.656	0.222
128	23.386	22.334	0.914
256	108.352	104.13	3.79

(b) array of four Transputers.

image side	Tt	Tp	Tc
16	0.075	0.044	0.024
32	0.3	0.232	0.054
64	1.371	1.167	0.17
128	6.321	5.586	0.602
256	28.851	26.04	2.288

(c) array of sixteen Transputers.

Table 6.8 Time of processing for the 2DFFT RC convolution implementation on several arrays.

Time is in secs.
Tt is the total time of processing.
Tp is the time of processing the forward and the inverse FFT.
Tc is the time of multiplication with the transformed subkernel.

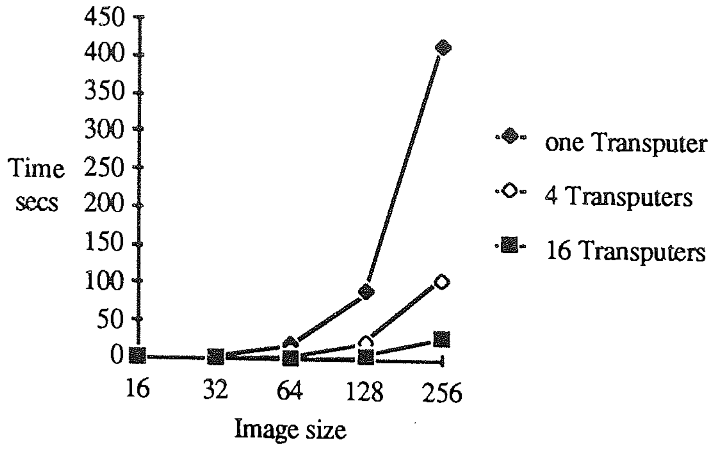
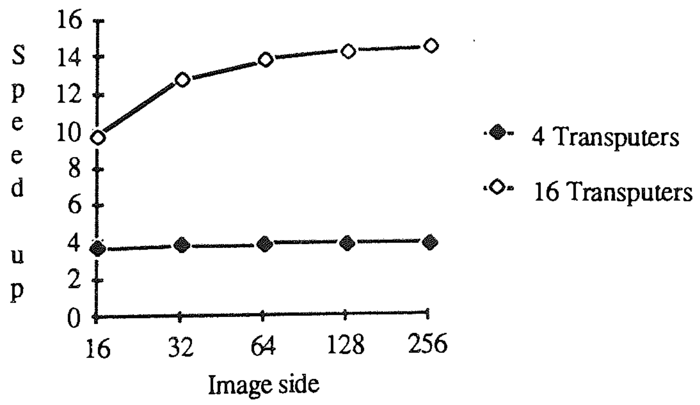


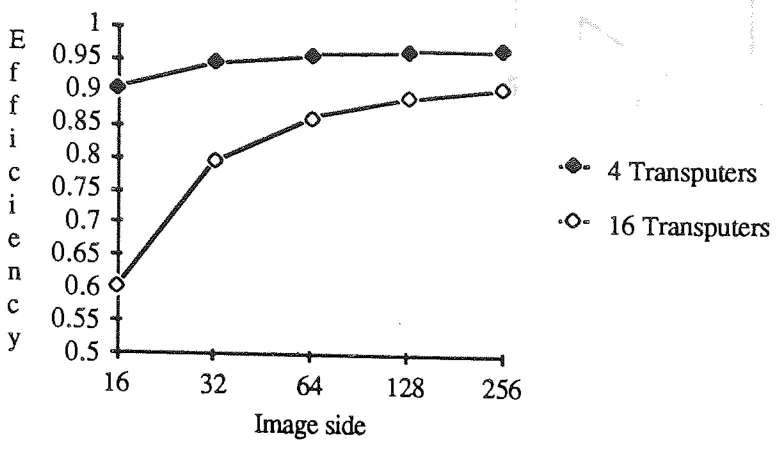
Figure 6.24 The time graph of the 2DFFT RC convolution implementation on array of a single, four, and sixteen Transputers.

The graphs show timings obtained from processing of the real-part of the data. Timings for the imaginary part are identical. The speed up and the efficiency have been calculated and their graphs are shown in figure 6.25.

It is clear from figure 6.25 that the speed up and the efficiency increase with the increase of the image size. The reason for this behaviour is due to decreasing of the communication overhead in the 2DFFT processing as discussed in section 6.5.6 and shown in figure 6.12. The efficiency for the array of four Transputers is better than that for the array of sixteen Transputers because the communication overhead is greater in the case of larger array sizes.



(a)



(b)

Figure 6.25 Performance graphs for 2DFFT RC convolution implementation array of four and sixteen Transputers.

6.7.2 Vector-Radix Based Implementation

This is the second method of convolution implementation using Fast Fourier Transform techniques. The procedure of the implementation is the same as described in section 6.1.2. Here the Vector Radix technique is used for the forward and the inverse Fourier Transform. The convenient parallel architecture proposed is the pyramid. This is because of the hierarchical structure of the Vector-Radix method provides a match with the pyramid architecture.

As mentioned in section 6.6.1 that there are two strategies for implementing the Vector-Radix method: the decimation in time which used a top-down scheme and the decimation in frequency which used a bottom-up scheme. For the convolution implementation either or both of these two strategies could be used. Figure 6.26 overleaf shows all possible architectures for the convolution implementation using a pyramid topology.

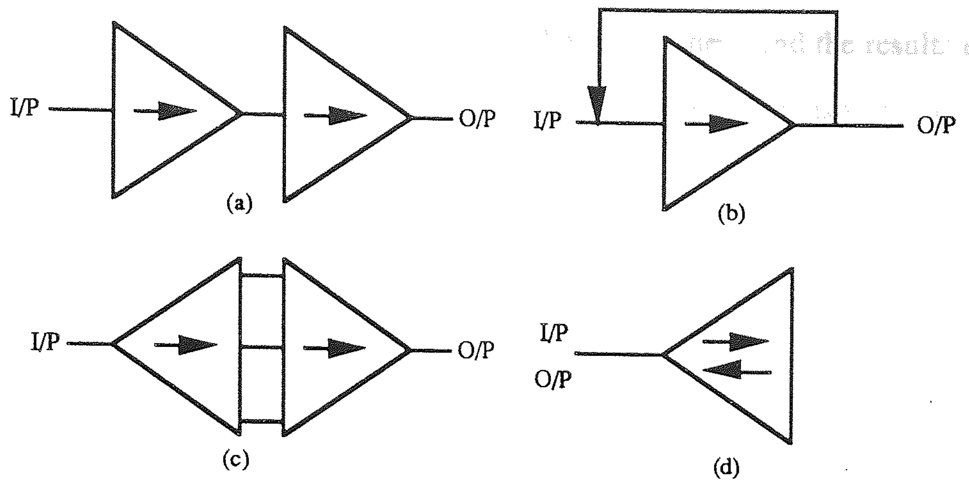


Figure 6.26 The possible configurations for 2DFFT VR convolution implementation using pyramid blocks.

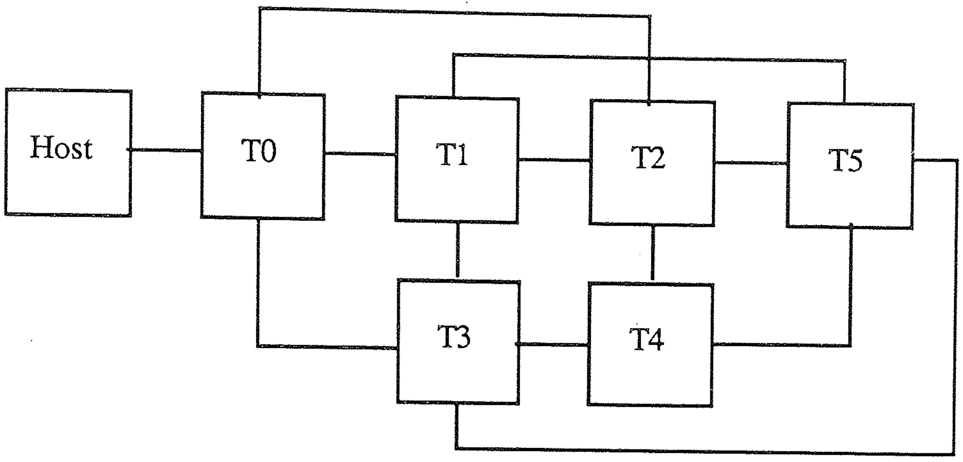
The triangle indicates a pyramid and arrows show direction of data flow.

It is known that the time spent for distributing data from one Transputer to several Transputers is larger than that for sending data between two equal groups of Transputers. It is then clear from figure 6.26 that the best efficiency can be achieved by the architectures in figure 6.26 (c) and (d) but the speed up for the architecture of (c) is more than that of (d) because there are more Transputers in (c) than that in (d). The architecture in figure 6.26 (d) may be used when there are not enough Transputers for implementing the architecture in figure 6.26 (c).

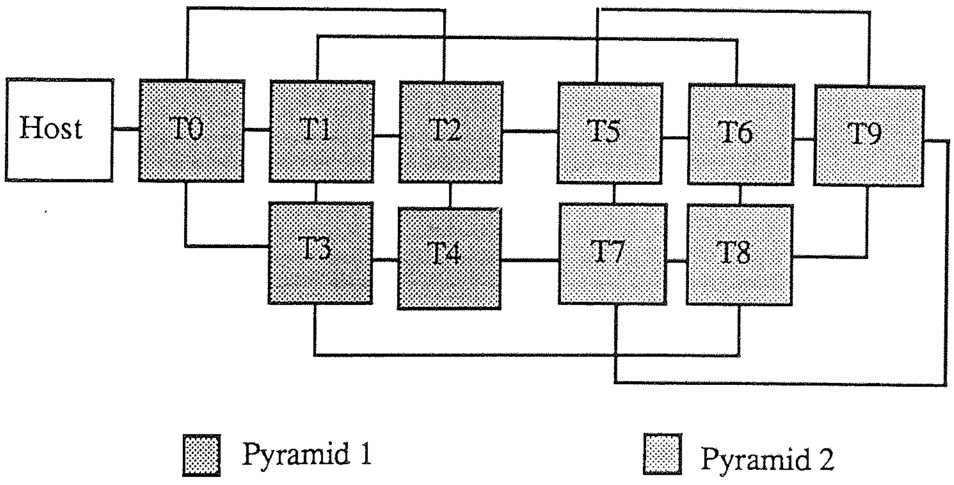
Two architectures based on figure 6.26 (c) have been proposed using pyramid of size four: one with common base level (CB) between the two pyramids (two CB pyramids) and the other with two separate pyramids (two-pyramid). The system block diagrams are shown in figure 6.27 (a) and (b) respectively.

The Occam model for each Transputer is the same as that shown in figure 6.8. Each Transputer in each level of the pyramid implements the same procedure, which is inputting the data from the previous level and process it according to the part of the algorithm to be processed in this level, and then send the results to the next level. So the program construct of the process P for each Transputers is

```
SEQ
        input data from the previous level
        implement the appropriate part of the algorithm for this
        level
        send the results to the next level
```



(a) two CB pyramids.



(b) two-pyramid

Figure 6.27 Two pyramid configurations for 2DFFT VR convolution implementation.

The time of processing for each stage of these two implementation is taken from Table 6.6, since the processing of convolution is two 2DFFT VR processing in cascade, and shown in Table 6.9.

image side	T11	T01+T02	T12
8	0.002	0.008	0.002
16	0.008	0.09	0.008
32	0.03	0.422	0.03
64	0.131	2.22	0.131
128	0.525	11.512	0.525
256	2.1	55.73	2.1

(a) for two CB pyramids configuration.

image side	T11	T01	T02	T12
8	0.002	0.004	0.004	0.002
16	0.008	0.045	0.045	0.008
32	0.03	0.211	0.211	0.03
64	0.131	1.11	1.11	0.131
128	0.525	5.756	5.756	0.525
256	2.1	27.865	27.865	2.1

(b) two-pyramid configuration.

Table 6.9 Estimated time of processing for implementation of 2DFFT VR convolution.

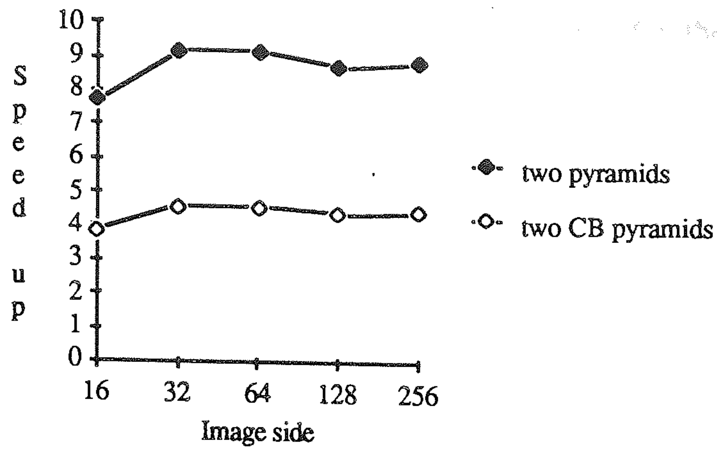
Time is in secs.
T11 is the time of processing at level one of pyramid one.
T01 is the time of processing at level zero of pyramid one.
T02 is the time of processing at level zero of pyramid two.
T12 is the time of processing at level one of pyramid two.

In these tables each column of time values belongs to one stage of the pipeline in the configuration. Better load balance has been achieved in the case of the two-pyramid configuration. This is because of the equal loads allocated to the two separate bases of the pyramids which represent two main stages of the pipeline (they contain 80% of the total number of Transputers in the network). The time of processing for each configuration is the time of that stage in the pipeline which takes the longest time. The time of convolution on a single Transputer is estimated by multiplying by two the time of processing of the 2D FFT VR on a single Transputer (taken from figure 6.19). Table 6.10 shows the time of processing for convolution on on a single Transputer and on two pyramid configurations. The performance graphs for these two implementations are calculated using Table 6.10 and are shown in figure 6.28 which follows the table.

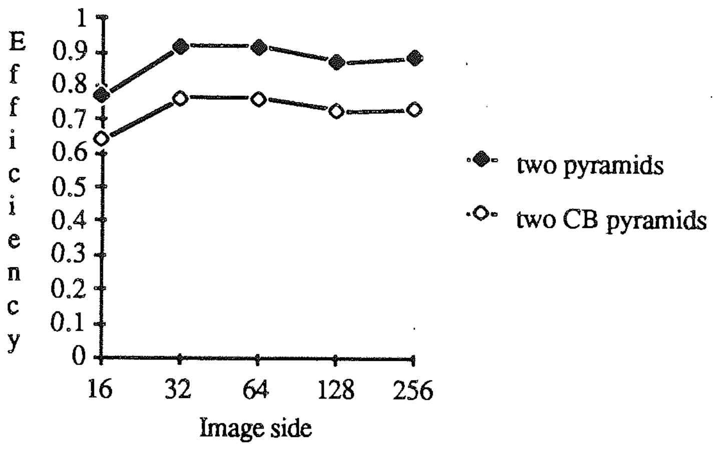
image side	one Transputer	two pyramids	CB pyramid
16	0.346	0.045	0.09
32	1.936	0.211	0.422
64	10.184	1.11	2.22
128	50.234	5.756	11.512
256	248	27.865	55.73

Table 6.10 Total time of processing of the convolution operation using 2D FFT VR method.

It can be seen from figure 6.28 that better performance, i.e. higher speed up and efficiency, has been achieved for the two-pyramid configuration. This is due to better load balance in this configuration as compared with that of the two CB pyramids, as was shown in Table 6.9.



(a)



(b)

Figure 6.28 Performance graphs for the two configurations: two CB pyramids and two pyramids.

6.8 Summary

This chapter has described three different parallel implementations of convolution to be used in context of digital image processing. These implementations are based on three different convolution methods: the direct method, the 2D FFT-based Row-Column (RC) method, and the 2D FFT-based Vector-Radix (VR) method. The array architecture has been chosen for implementation of the direct method and the Row-Column method; and the pyramid architecture for the Vector-Radix method. Parallel implementations for both 2D FFT-based methods are novel.

The direct method shows a good performance for convolution with small kernel sizes. For example, for image size of 128x128, kernel size of 11x11 and an array of 16 Transputers the time of processing was 3.89 seconds and speed-up factor of 14.6 was achieved; for kernel size of 15x15 the speed-up was increased to 15.1 and the time of processing was 6.932 seconds. For larger kernel sizes the FFT-based techniques are similar or better since they involve overall smaller number of operations.

It was interesting to compare performance of two different FFT-based methods of convolution in which performance does not depend on the size of a kernel. Both implementations showed approximately similar performance as measured by the efficiency factor; for example for image size of 128x128 this factor reaches value of about 0.89 for the RC-based implementation on an array of 16 Transputers and 0.872 for the VR-based implementation on two pyramids with 10 Transputers. The overall execution time was lower for the VR method than for the RC method; for example, the processing time for image size of 128x128 using VR method (10 Transputers) was 5.756 seconds while for the RC method (16 Transputers) it was 6.321 seconds. This was to be expected since the number of operations is lower in the VR algorithm.

In the VR implementation the efficiency increases with the increase in the number of Transputers because no intercommunication is needed within a single level of the pyramid. This is a very important advantage. One drawback of this implementation is that the performance decreases with the increase of the image size (this is inherent in the algorithm itself which gives the best load balance for image of size 64x64 as shown in figure 6.17). This, however, is not very serious as the decrease is very small. The second drawback is that this method requires a large amount of memory. The memory requirement is again a consequence of using the particular algorithm; as the amount of data does not decrease with each iteration in the algorithm its parallel implementation needs the same amount of memory on each level of the pyramid.

The RC-based implementation can be run without change on an array of Transputers of any size and its good load balance will be preserved for any size of the network. It requires less memory than the VR method. For example, the memory size required for the RC method on an array of 4 Transputers is half of that required for the VR method on the pyramid of size four. The main drawback of the RC method is that the efficiency of this method falls with the increase in the number of Transputers in the network. This is caused by the increase in the communication overhead (Coh) brought by the matrix transposition operation and is especially apparent for larger networks. However, the decrease in the efficiency measure is not very large, especially for large image sizes, because a lot of effort has been put into the design and development of an efficient communication protocol.

In conclusion the 2D FFT VR-based implementation of convolution shows the best performance as compared with two other implementations, both on a single Transputer and on a network of Transputers. The main drawback of this implementation, the memory requirement, should become less significant as memory becomes cheaper and more compact. This implementation is therefore suitable for inclusion in a larger system which includes several convolution operations, such as a Granlund method which will be described in the next chapter.

Chapter 7

Texture Segmentation by Granlund Method

The Granlund method, outlined in section 2.3.2, is one of the effective methods of texture segmentation. It relies on detecting dominant frequencies and directions within spatial windows whose size increases with decreasing levels of resolution. This is achieved by convolving an image with a kernel having a desired response. A number of convolution 'blocks' are combined together in a hierarchical structure to deal with different resolution levels. The convolution block is the most computationaly demanding part of the algorithm. As the Granlund method is a successful texture segmentation scheme, it seemed appropriate to attempt its implementation as an example of the application of convolution. The full parallel implementation of the method has not been achieved due to insufficient number of Transputers being available to the department during this project. However, its main part, the convolution, has been implemented in parallel as described in the previous chapter. A system-level design for the complete method has been proposed and performance of the full algorithm has been estimated. Although not part of the mainstream of the final thesis the design and evaluation of the convolution kernel was an important undertaking in course of the work. It is therefore described fully in Appendix A.

7.1 General Picture Processing Operator

A general picture processing operator proposed by Granlund (1978, 1980b) is capable of detecting and describing structure as opposed to uniformity within local regions of an image. The basic function of the operator is to determine a vector for each pixel value of the image. The magnitude of this vector represents the feature value, and its phase represents the direction of that feature. The feature could be the step size of the edge, density of line, or the amplitude of the grey scale variation of the texture. This means that the operator gives a zero output for a uniform region, and non zero output for a structured region.

Granlund chose a Fourier transform to analyse and design the operator. This is because the Fourier transform describes the relationship between the spatial and frequency domain. Also, it can be useful for the design of the operator because it is more meaningful to specify the frequency response of the operator than the impulse response.

A local Fourier Transform of an image can be defined as the Fourier Transform of the image inside a small window. If the image inside the window is close to 1-dimensional then the Fourier transform of it, which represents the energy, will be concentrated in a narrow sector oriented at the same angle as the gradients inside the window (Granlund, 1980a). The sector is narrower when the window is more linear. Figure 7.1 shows the Fourier Transform of the image within a window. It can be seen from figure 7.1 that it is quite useful to represent the image in the frequency domain by polar coordinates.

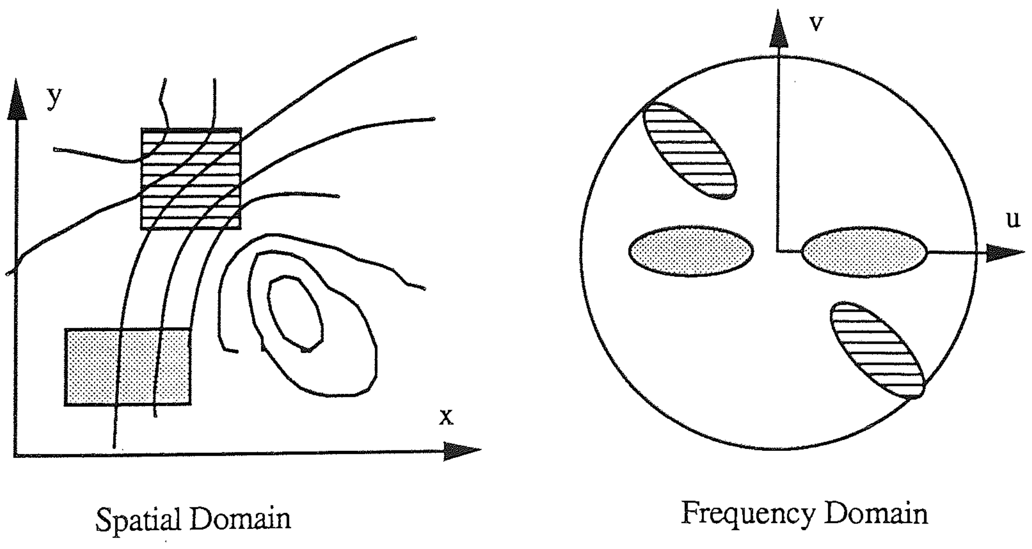


Figure 7.1 Local Fourier Transform.

Let f(x, y) be the input image in the spatial domain, and $F_c(u, v)$ is the Fourier Transform of this image in the Fourier domain using cartesian coordinates.

$$F_c(u, v) = \sum_{u=u_1}^{u_2} \sum_{v=v_1}^{v_2} f(x, y)\, W_N^{(ux+vy)},$$

where $W_N = EXP(-2\pi j/N)$, and N is the size of each dimension of the image. Let P_{θ_n} be the magnitude of the transformed image within a window of a certain size for a direction θ_n

$$P_{\theta_n} = \sum_{r=r_1}^{r_n} \left| F_\rho\left(r, \theta_n\right) \right|,$$

where $F_\rho(r, \theta_n)$ is the polar coordinates representation of $F_c(u, v)$,

$$r = \sqrt{u^2 + v^2} \ ,$$

and

$$\theta = \tan^{-1}\left(\frac{v}{u}\right) .$$

The magnitude of the resulting vector for f(x, y) will be obtained by finding the maximum value of P_{θ_n} over a number of directions $\max_\theta\{P_{\theta_n}\}$; its phase will depend on θ_n for which P_{θ_n} makes its maximum value.

The above description has shown that the local Fourier Transform is useful for calculating the maximum variation of the image grey levels within a certain window size. The next step is to specify the type of the window and to show that the result can be found by convolving the image with the window function.

According to the uncertainty principle (see Appendix A section A.4) it is optimal to use the Gaussian window function (Granlund, 1978) in order to get the minimum value of the uncertainty formula,

$$g(x, y) = EXP\left[-\alpha\left(x^2+y^2\right)\right].$$

To realize window functions with a response centred at a certain frequency other than zero, a complex exponential function is multiplied with the Gaussian one,

$$w(x, y) = g(x, y) \, \mathrm{EXP}[2\pi j \, (u_o x + v_o y)], \qquad (7.1)$$

where (u_o, v_o) is the centre frequency of the frequency response of $g(x, y)$. The window function $w(x, y)$ is a complex vector and it is called the analytic function. The filters should be of the real type. To construct such filters from the analytic function, the real part and the imaginary part are considered as separate filters:

$$w(x, y) = w_e(x, y) + j \, w_o(x, y).$$

Subscripts e and o are for even and odd functions respectively because normally $w(x, y)$ is hermitian, i. e., it has even real part and odd imaginary part.

A set of window functions is used; each of the functions is different from the other by its direction angle. The first step in obtaining the required image transform is to convolve these window functions with the original image $f(x, y)$ to get

$$A(r, \theta_n, x, y) = f(x, y) \, w(r, \theta_n, x, y), \qquad (7.2)$$

for $\theta_n = \theta_1, \ldots, \theta_k$, where r is the centre frequency of the window function w, and k is the number of window functions used. The final output, which is called the first transform, is achieved by taking the maximum value of A and multiplying it by the complex exponential of the corresponding direction,

$$f_r(x, y) = \max \left| A(r, \theta_{n_{max}}, x, y) \right| e^{j\theta_{n_{max}}}.$$

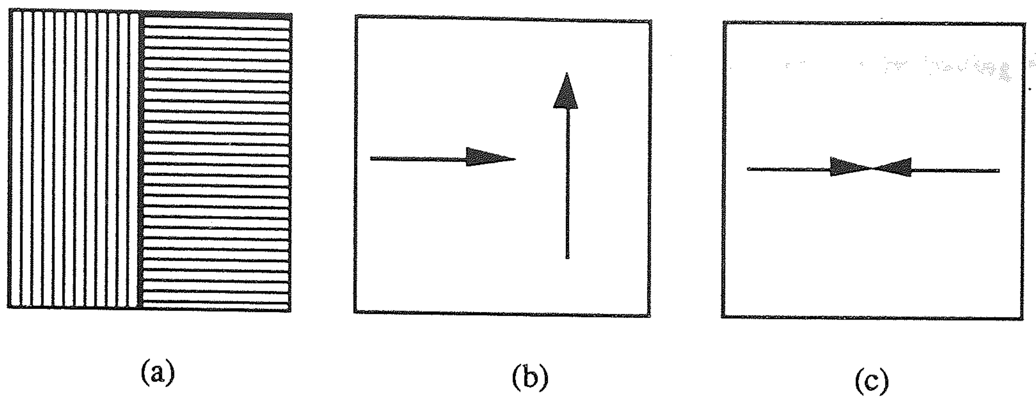


Figure 7.2 Transformation process. (a) original image (b) first transformation. (c) second transformation.

Figures 7.2 (a) and (b) show an input image and its transformation using the above procedure. Since it is better to represent the maximum difference by vectors with opposite directions, $\theta_{n\max}$ is multiplied by a factor of two to achieve this requirement,

$$f_r^{(1)}(x, y) = \max\left|A(r,\theta_n, x, y)\right| e^{j2\theta_{n_{\max}}}. \qquad (7.3)$$

Figure 7.2 (c) shows the effect of this change to the previous operation.

7.2 Hierarchical Structure

It has been shown that the analysis of the local region of varying size of the picture led to the extraction of useful information. In each transformation stage a certain window size is used to give the information within a limited frequency band. It is quite useful that the windows become increasingly larger for higher transformations. This is because of the phenomenon that in the higher level only high level global information remains, and hence the size of the operational region or the window must be increased.

The hierarchical structure is shown in figure 7.3 below. The combination of several levels of transformations is used to cover the whole frequency range starting with the highest central frequency in the lowest level and consequently having the smallest window size. In higher levels the centre frequency decreases with decreasing bandwidth.

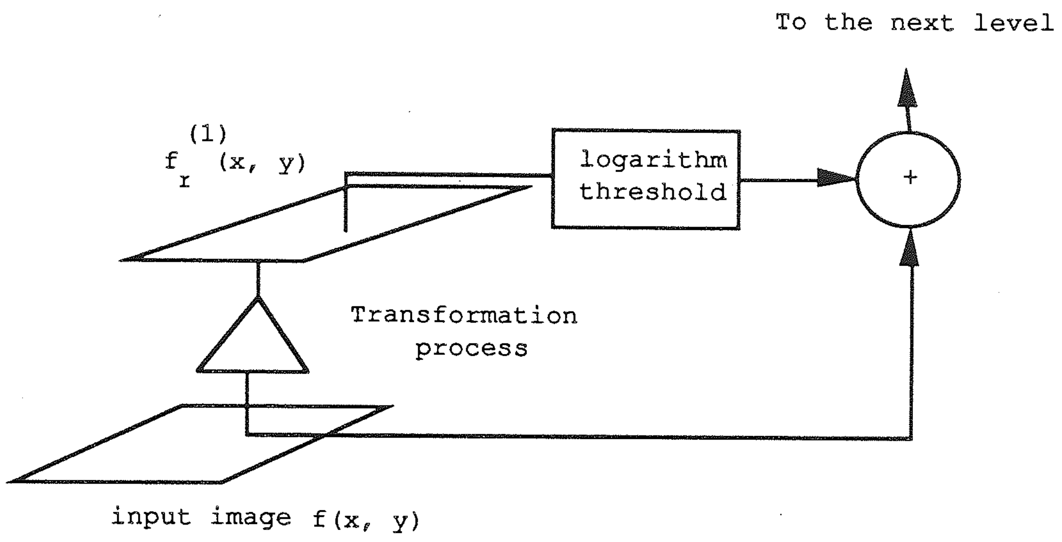


Figure 7.3 Hierarchical structure of Granlund method.

It was found (Granlund, 1978) that the best result is obtained if the function $f_r^{(1)}$ is transformed by applying the following procedure

$$f_r^{'(1)}(x, y) = \begin{cases} f_r^{''(1)}(x, y)\, e^{j2\theta} & f_r^{''(1)}(x, y) < 0 \\ 0 & f_r^{''(1)}(x, y) \geq 0 \end{cases}, \tag{7.4}$$

where,

$$f_r^{''(1)}(x, y) = k\left[\log\left|f_r^{(1)}(x, y)\right| - C_{th}\right],$$

where k is a proportionality factor and C_{th} is a constant to set a bias level.

This procedure removes low-level noise by thresholding and gives a compression of the range of values of $f_r^{(1)}(x, y)$, especially on the middle amplitude range, by taking the logarithm.

This rescaled function is now to be combined with the original image by an addition operator (+) to form a picture function

$$d_{r_1}^{(1)}(x, y) = \left[\left| f_r^{'(1)}(x, y) \right| + f(x, y) \right] e^{j2\theta} . \tag{7.5}$$

In the next level a window of larger size and lower centre frequency is used to obtain information within a lower frequency range. For actual implementation the window size stays constant in the higher levels, but since the image size decreases in the higher levels, the relative size of the window with respect to the image size increases (Granlund, 1978).
The addition operation with the logarithmic function means multiplication. The necessity of using the original image in each transformation level is because of loss of some information in each level. By the combination of the transformed image with the original image this information is retained.

7.3 Line and Edge Operators

The requirements for the 'general picture processing operator' have been laid out in the previous section. This section briefly outlines the design of a convolution kernel with the direct response. Theoretical background relevant to filter design is given more fully in Appendix A. There are two requirements which should be followed to to choose the desired function for the line and edge operators:

a) The degree of local concentration in the two domains, the spatial and the frequency, should be high. This could be achieved by applying the uncertainty principle and trying to get a function which gives the smallest value of this relation. The family of Gaussian type functions are good candidates.

b) The assumption is that the good neighbourhood orientation estimates can be obtained for a 1-dimensional neighbourhood by simple combination of outputs from different operators. These estimates should be invariant with respect to the frequency content (Tasto & Wintz, 1971). This can be achieved by functions with frequency response of separable type in the polar coordinates,

$$F(\rho, \theta) = F1(\rho)\ F2(\theta), \quad \text{for } -1<\rho\leq 1 \text{ and } 0\leq\theta<\pi$$

where ρ is the frequency and θ is the angle.

From the above discussion it seems that the Gaussian function satisfies the first requirement but fails to satisfy the second one. That prompted Granlund and colleagues (Knutsson *et al.*, 1980) to define another function which satisfies the above requirements. This function is

$$F(\rho, \theta) = F_e(\rho, \theta) + j\ F_o(\rho, \theta), \tag{7.6}$$

where,

$$F_e(\rho, \theta) = v(\rho)\ v_e(\theta),$$

$$F_o(\rho, \theta) = v(\rho)\ v_o(\theta),$$

$$v(\rho) = \mathrm{EXP}\left[\frac{-4}{\log_{10} 2} B^{-2} \log_2^2\left(\frac{\rho}{\rho_i}\right)\right],$$

$$v_e(\theta) = \cos^{2A}(\theta - \theta_k), \qquad A=1,\ 2,\ \ldots\ldots$$

$$v_o(\theta) = v_e(\theta)\ \mathrm{sign}\left[\cos(\theta - \theta_k)\right],$$

ρ_i = centre frequency,

B = 6 dB sensitivity bandwidth in octaves,

θ_k = orientation angle,

A = angle selectivity.

B is defined by the following equation,

$$B = 2\log\left(\frac{\rho_u}{\rho_l}\right),$$

where ρ_u and ρ_l are the upper and lower cutoff frequencies respectively.

The function $F(\rho, \theta)$ in equation (7.6) represents a complex ideal frequency response. There is no way to approximate the complex function because all the approximation methods are applied to the real functions; however, each part of this complex function can be assumed to be a separate frequency response. In this case the real even part represents the frequency response of the line operator and the imaginary odd part represents the edge operator. Any of the approximation methods described in Appendix A section A.3 can be applied for each one separately to get the approximated impulse response for each of them.

For the design of the kernels the window method was used and a Fortran program was written to implement this design. The program contains three parts. The first part is to generate the required analytic function in the frequency domain. The second part is to take the inverse Fourier transform of the function to get the analytic function in the spatial domain. The origin of this analytic function is placed at the centre of 2-dimensional array. The final step is to truncate the analytic function in the spatial domain to the required kernel size around the centre. The Fortran program was run on the VAX/CLUSTER mainframe computer. Each set of kernels of certain frequency was saved in a file and was transferred to the IBM PC. Since the values of the kernels are of real type an Occam procedure had to be written to read the file of the kernel set because there was no such procedure supplied with the development system.

Figures 7.4 (a)-(f) on pages 168-173 show the frequency and impulse response of the generated line and edge operators.

7.4 System Implementation

For system implementation, both operators are applied at the same time to the image; the edge operator detects edges which are transformed into line and the line operator detects lines. The theory implies that all the directional operators are applied at each level and the output is that of the maximum value. For example assume that the line and edge operators are L(i, k) and E(i, k), where i denotes the operator number and k is the coefficient number of that operator. Usually the operators within the level are of the same frequency but with different directions, so the values of i are from 1 to I, where I is the number of the operators used in this level and is usually equal to eight; consequently the directions are from 0 up to ((I-1) * π/4). If K is the number of the coefficients of each operator, then the values of k range from 1 to K. For applying the operators of each level to the input image, the convolution operations for each pixel of the image is as follows:

Assume that the real part and the imaginary part of window of the image centred at position (t, u) are x(k) and y(k) respectively, where k = 1, ..., K. Then there are four values to be calculated and these are

$$f_1(i) = \sum_{k=1}^{K} L(i, k)\, x(k),$$

$$f_2(i) = \sum_{k=1}^{K} L(i, k)\, y(k),$$

$$f_3(i) = \sum_{k=1}^{K} E(i, k)\, x(k),$$

$$f_4(i) = \sum_{k=1}^{K} E(i, k)\, y(k).$$

Then the amplitude content in direction i is Z(i), and

$$Z(i) = \sqrt{\sum_{j=1}^{4} f_j^2(i)} \ .$$

To get the final output for this pixel value in the output image A(t, u) of the current level, the maximum value of Z is used since this gives the indication of the maximum grey level variation in certain direction. The output A(t, u) is

$$A(t, u) = Z_{max}\, e^{(r-1)\frac{\pi}{4}},$$

where

$$Z_{max} = \max Z(i) \qquad i = 1, \ldots I,$$

and r is the operator number which gives the maximum value, i.e. $Z(r) = Z_{max}$.

The system consists of several levels. The processing operations in each level are similar. The operation is convolution, where the image is convolved with the operator function (window function); this is then followed by logarithm and addition operations. The block diagram of system which consists of three levels is shown in figure 7.5, the block called C is the convolution block.

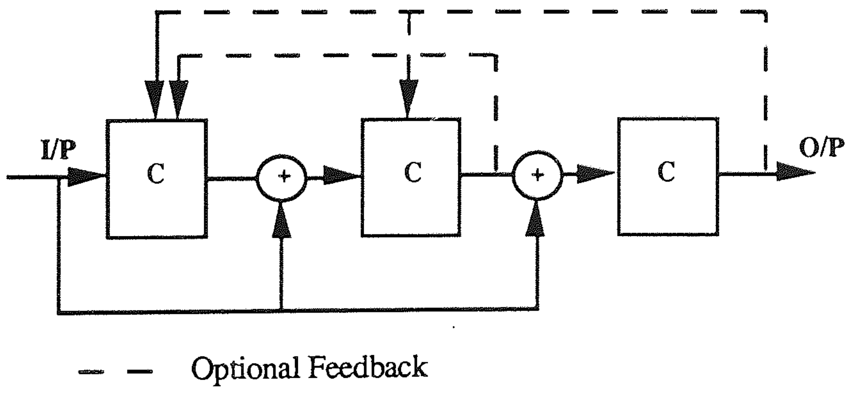


Figure 7.5 Granlund method: System block diagram.

Two sets of kernels have been designed using the window method; each set differs in the centre frequency . The size of the region of support in the frequency domain used for the design is 64x64. The centre frequency for the first set is 0.555 with kernel size 5x5 while it is 0.37 with kernel size 7x7 for the second set (the centre frequency values were chosen on the basis of experimentation). Each set of kernels are eight line operators and eight edge operators with different orientation angles,

$$\theta=k*22.5 \qquad \text{for } k=0, 1, .., 7$$

The kernels of the first set are applied in the first level of transformation while the kernels of the second set are applied in the second level of transformation. Two images of size 64x64 are tested on a single Transputer and the results of these test are shown in figure 7.6 on page 174. The results show that in the first transformation regions with different texture are separated and in the second transformation borders between these regions are clearly marked.

7.5 Performance Estimation

The performance of the complete parallel implementation of the Granlund method has been estimated on the basis of the actual performance of the convolution block. The convolution block chosen for this purpose is the 2DFFT VR because its performance for large kernel size is better than that of the 2DFFT RC convolution block or the direct convolution block. The 2D FFT VR convolution block has been implemented on a pyramid architecture which was described in section 6.7.2. The estimation of the Granlund method has been done for a two-level system, capable of finding borders between regions of different texture. Image resolution on the higher level of the system is four times lower than resolution on the lower level. This indicates that processing time is longer on the first level. It would therefore be inefficient to use on the second level a convolution pyramid of the same size as that on the first level. A better load balance can be achieved by using a smaller pyramid on the higher level. Based on these facts the following performance estimation can be worked out.

Assume that the block diagram of figure 6.27 (b) on page 146 is used as a convolution block where two separate pyramids are used for the first transformation level of the system shown in figure 7.5. In the second transformation level the block shown in figure 6.27 (a), with a common base two pyramids, is used as a convolution block. To estimate the performance of this proposed system the following estimated times are used. The time of processing on a single Transputer (ts) is taken from figure 6.19 and given here again in table 7.1. The times of processing on level one (t1) and two (t2) are taken from tables 6.9 (a) and 6.9 (b) respectively and given again in table 7.2. The estimated time of processing on a single Transputer and on the network of Transputers are shown in table 7.3.

image side	1 Transputer
16	0.185
32	1.013
64	5.274
128	25.835

Table 7.1 Time of processing of convolution on a single Transputer in the Granlund method.

Time is given in secs.

image side	T11	T01+T02	T12
8	0.002	0.008	0.002
16	0.008	0.09	0.008
32	0.03	0.422	0.03
64	0.131	2.22	0.131
128	0.525	11.512	0.525
256	2.1	55.73	2.1

(a) for two CB pyramids configuration.

image side	T11	T01	T02	T12
8	0.002	0.004	0.004	0.002
16	0.008	0.045	0.045	0.008
32	0.03	0.211	0.211	0.03
64	0.131	1.11	1.11	0.131
128	0.525	5.756	5.756	0.525
256	2.1	27.865	27.865	2.1

(b) two pyramids configuration.

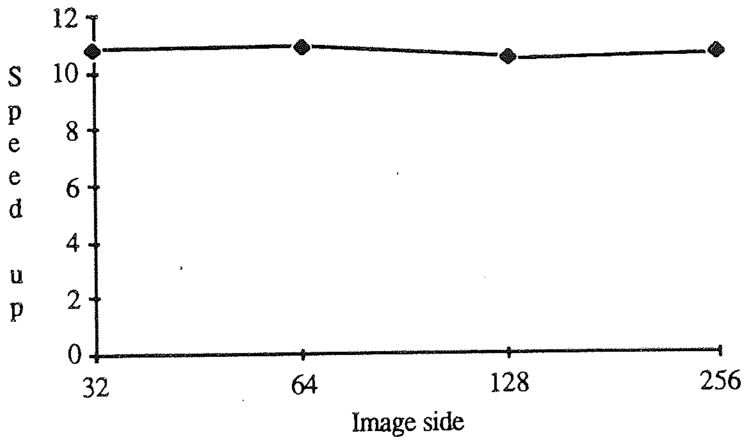
Table 7.2 Estimated time of processing for implementation of 2DFFT VR convolution.

Time is given in secs.

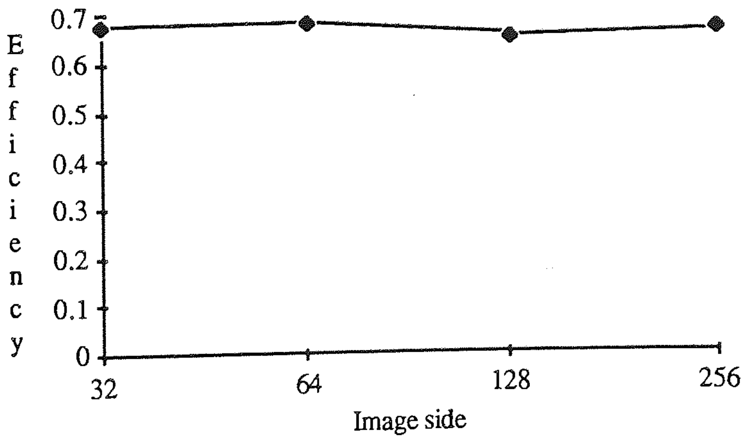
image side	ts	t1	t2
32	2.282	0.211	0.09
64	12.12	1.11	0.422
128	60.418	5.756	2.22
256	298.234	27.865	11.512

Table 7.3 Time of processing for two-level Granlund system.

Figures 7.7 (a)-(b) show the speed up and the efficiency graphs for the two-level Granlund system. As seen from these graphs the performance is poorer than that of the convolution block itself (compared with figure 6.28 on page 149). This is because the load balance is not very good as shown in table 7.3. This drawback can be eliminated by using other sizes of pyramids to get better load balance.



(a)



(b)

Figure 7.7 Performance graphs for the two-level Granlund system.

7.6 Summary

In this chapter the Granlund method of texture segmentation has been described and analysed. Two main implementation issues have been discussed in particular; the design of the convolution kernels and the design of a suitable parallel architecture. Out of the two methods of the kernel design, the window method and the least square method, the window method was used because of its simplicity. This method produced satisfactory results. The choice of the centre frequency and other parameters for kernel design is a very complex issue and strongly depends on the type of the images under processing; further research is required in this direction.

The most computationally intensive operation in Granlund method is convolution. It has been proposed to use the Vector-Radix method of convolution and its implementation on the pyramid architecture because this implementation showed performance superior to both the Row-Column method and the direct method. Performance of the Granlund system with two levels has been estimated based on the results described in chapter six.

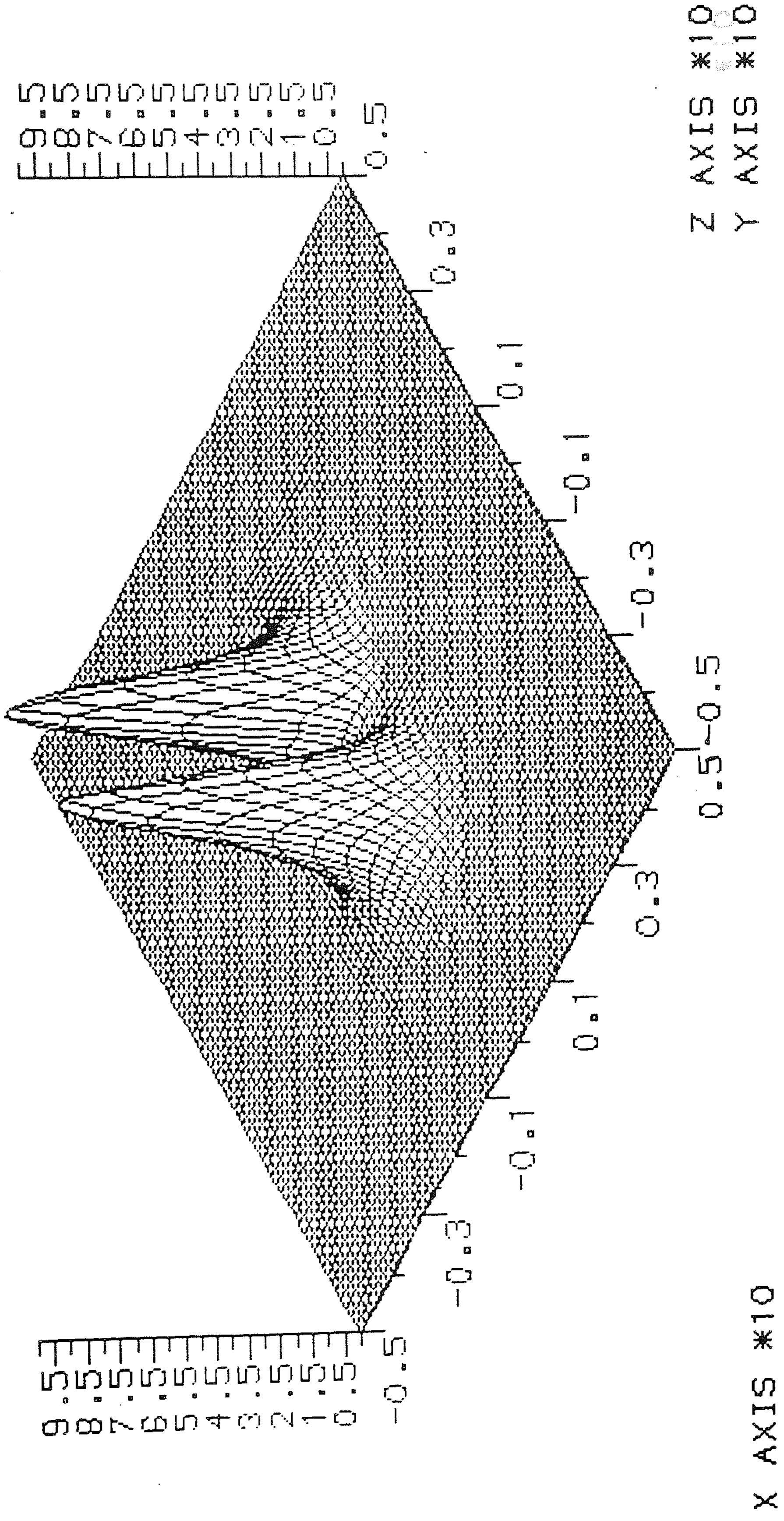


Figure 7.4 (a) Ideal frequency response of the Line operator.

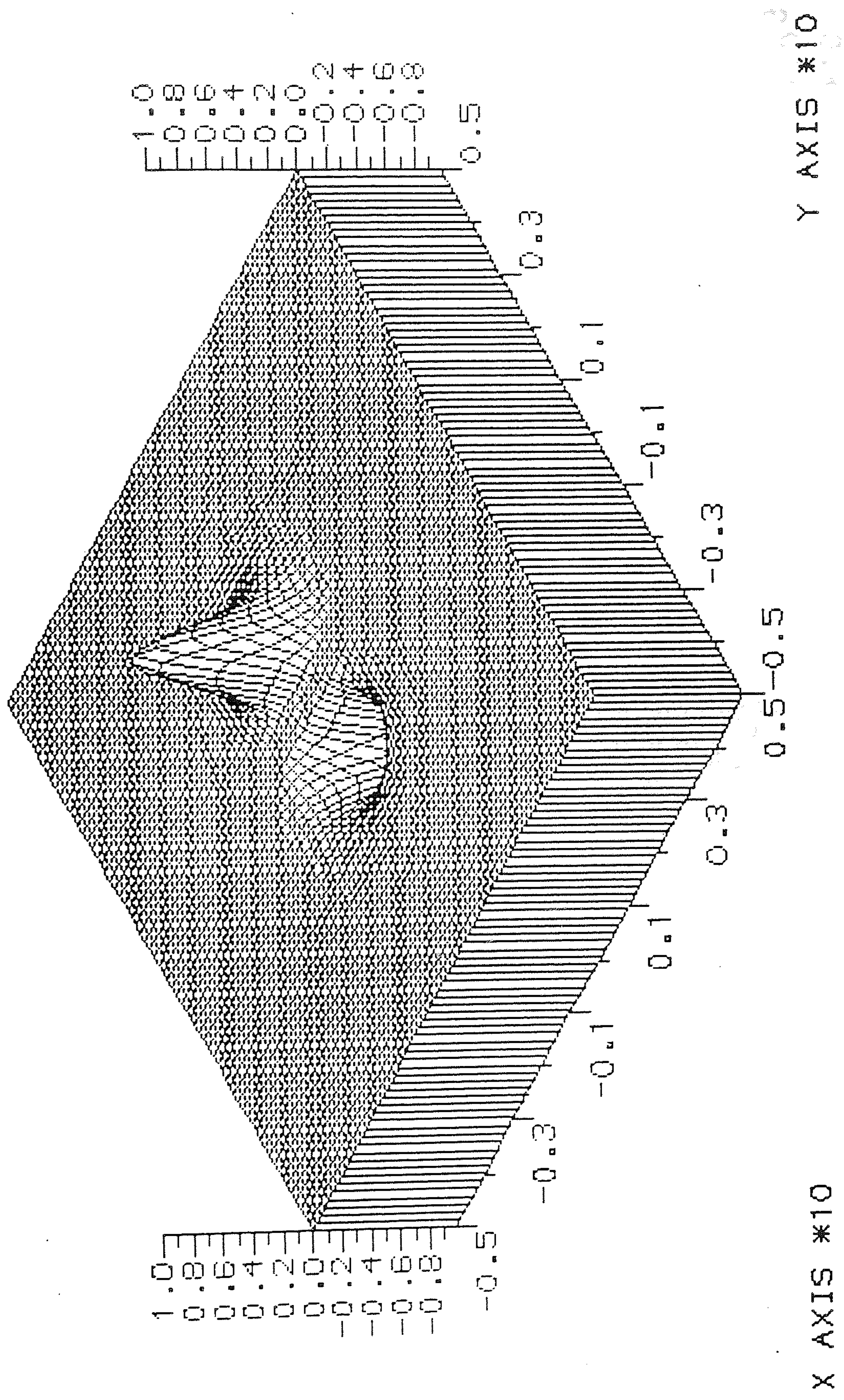


Figure 7.4 (b) Ideal frequency response of the Edge operator.

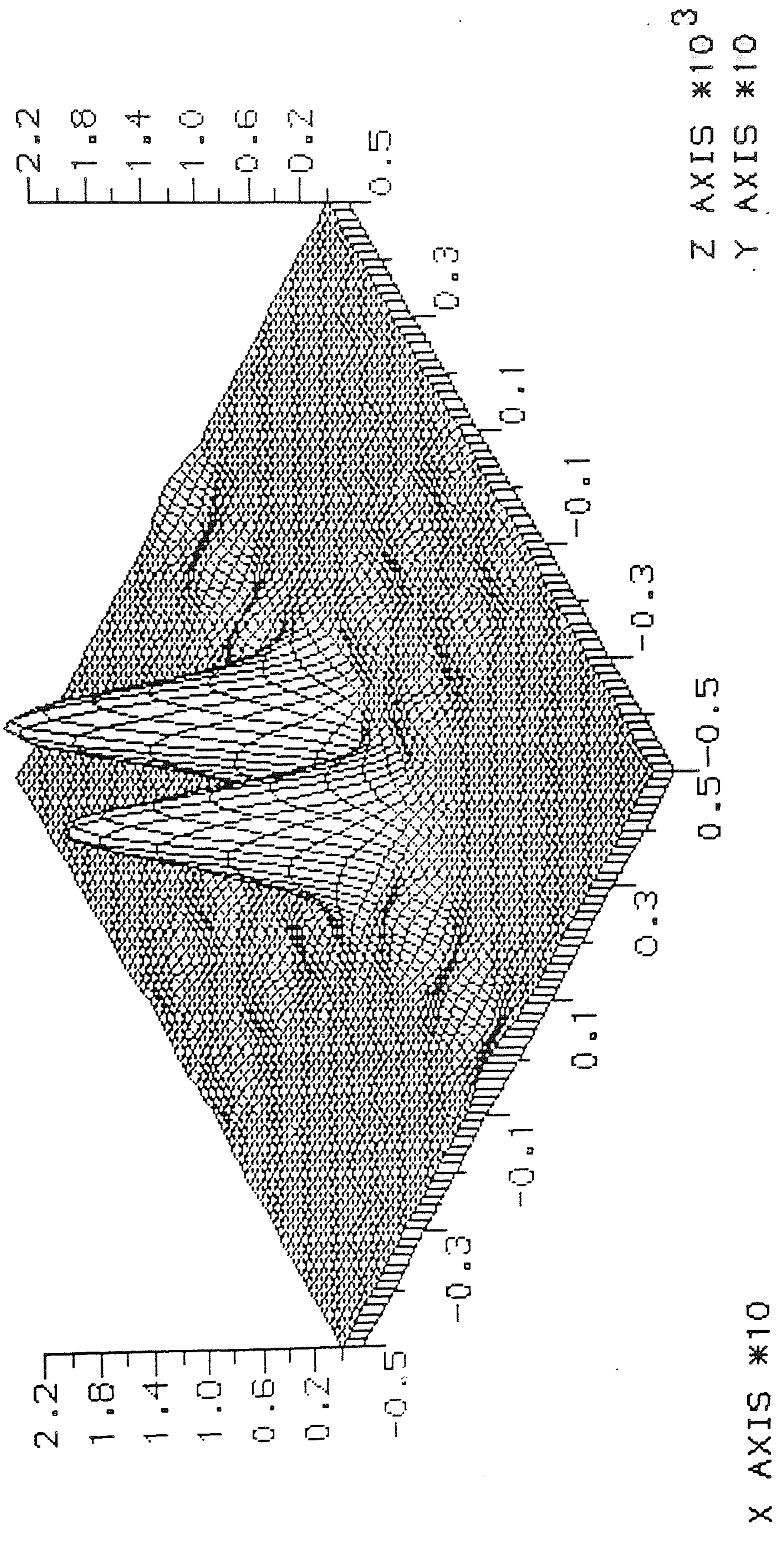


Figure 7.4 (c) Approximated frequency response of the Line operator.

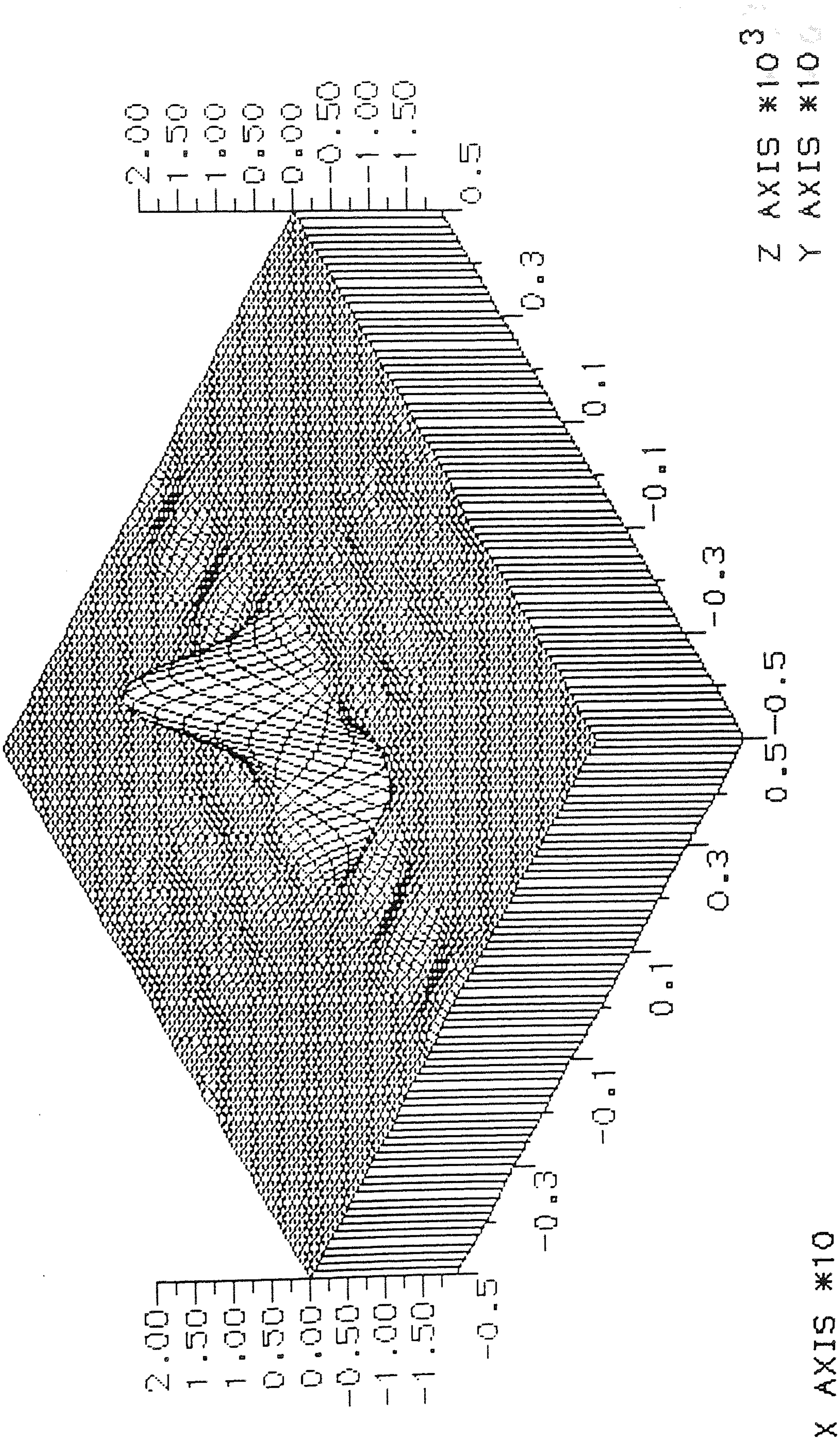


Figure 7.4 (d) Approximated frequency response of the Edge operator.

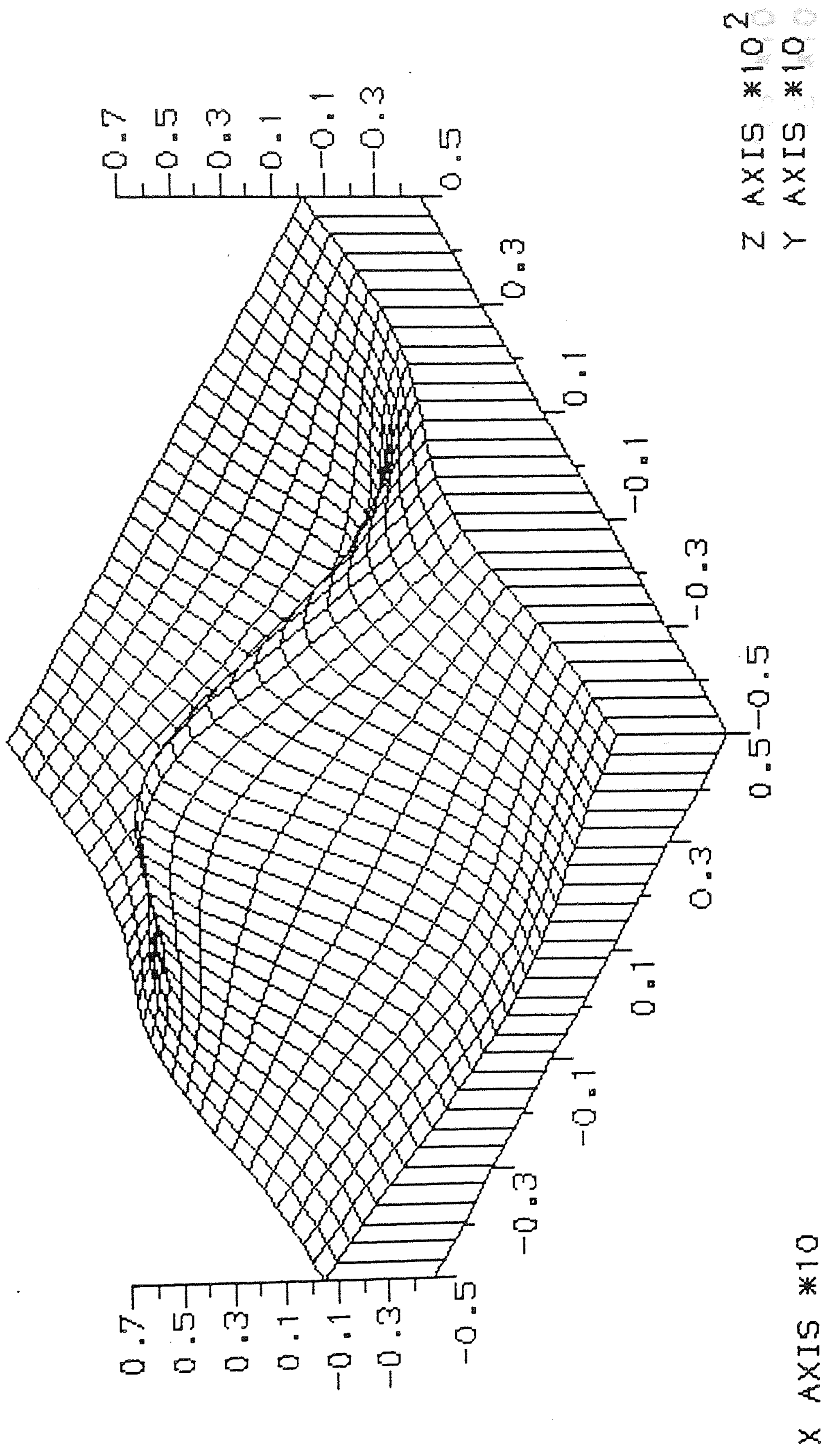


Figure 7.4 (e) Impulse response of the Line operator.

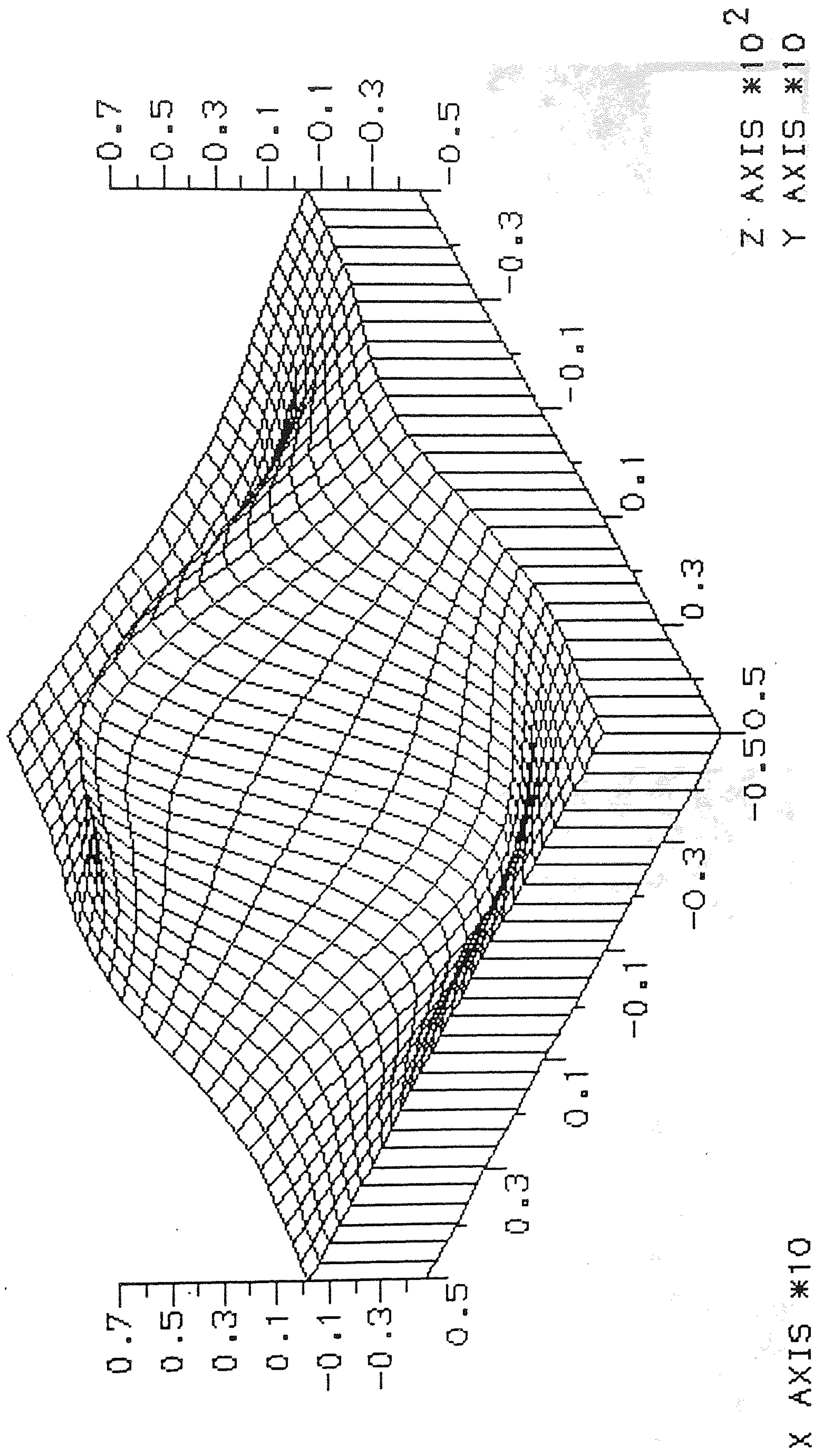
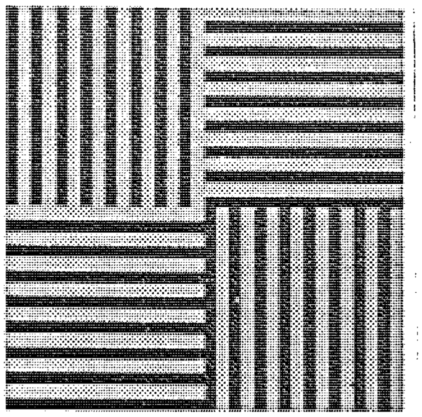
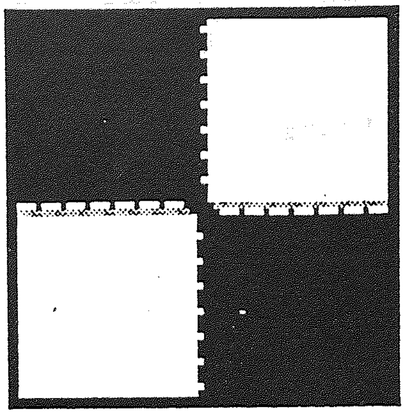
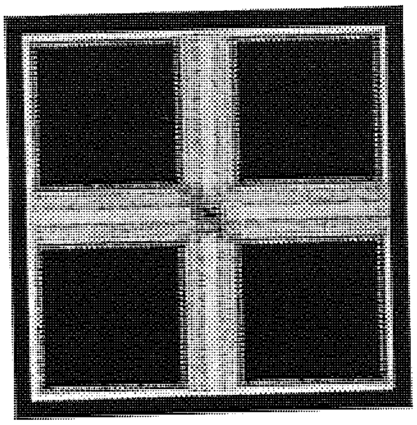
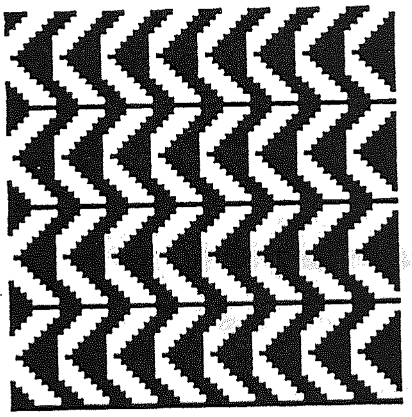


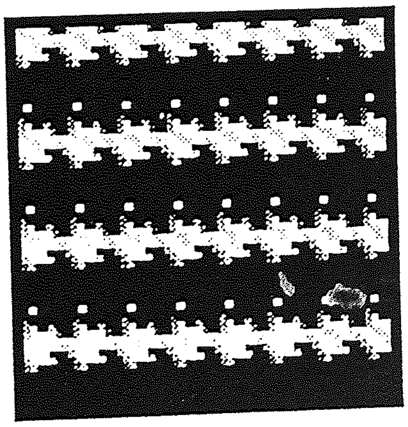
Figure 7.4 (f) Impulse response of the Edge operator.


(a) Test image one


(b) 1st transform of (a)


(c) 2nd transform of (a)


(d) Test image two


(e) 1st transform of (d)

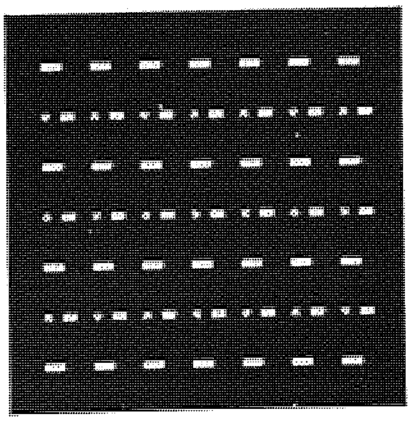

(f) 2nd transform of (d)

Figure 7.6 Two test images and their transforms using Granlund method.

Chapter 8

Region Segmentation by Split and Merge Algorithm

It has been mentioned in chapter two that one of the more efficient method for region segmentation is the Split and Merge algorithm (S & M), due to its efficiency for segmenting the image into homogeneous regions. It is also an example of a quadratic regular decomposition, a strategy widely used in image computing. The purpose of this chapter is to describe the implementation of the Split and Merge algorithm for image segmentation using the spatial parallelism technique.

The Split and Merge algorithm uses a quadtree as its basic data structure and this is reflected by the choice of the hierarchical pyramid as an underlying hardware architecture. On the base level of the pyramid each processing element (a Transputer) is provided with a sub-image which is a $1/4^N$ portion of the whole image, where 4^N is the size of the pyramid. Processes are invoked in parallel to perform the segmentation of the sub-images. The results of local segmentation are exchanged between four adjacent processors which have a common parent node, and thus the final segmentation is established for a given level. These results can be then passed to a higher level of the pyramid and the same segmentation routine used, until the top of the pyramid is reached, i.e. the final segmentation of the whole image achieved. The performance measures are given for the implementation of the algorithm on a pyramid of Transputers.

8.1 Split and Merge Algorithm

The Split and Merge algorithm described by Horowitz & Pavlidis (1976) segments an image into regions which are uniform in some predefined sense. Formally, region uniformity may be defined by a predicate. A segmentation process can be described as follows.

Let R be the domain of an image and let f(x, y) be the brightness function defined on R. A logical predicate P is defined on subsets S of R as follows,

$$P(S) = \left\{ \begin{array}{ll} \text{TRUE} & \begin{array}{c}\text{if there exists a constant (e)} \\ \text{such that } |f(x_1, y_1) - f(x_2, y_2)| \leq e \\ \text{for any two points in S}\end{array} \\ \text{FALSE} & \text{otherwise} \end{array} \right\}$$

where e is a prescribed error tolerance. A segmentation of R can be defined as the partition of R into subsets S_i, i = 1 ... m, for some m such that:

a) $R = \bigcup_{i=1}^{m} S_i$

b) $S_i \cap S_j = \emptyset$ for all $i \neq j$

c) $P(S_i)$ = True for all i

d) $P(S_i \cup S_j)$ = False for all $i \neq j$ and subsets S_i, Sj are adjacent.

There are three main implementation schemes, which differ in the form of the initial partition. In a pure merging scheme, the algorithm starts with a partition satisfying (c) and proceeds to fulfil (d). A pure splitting scheme starts with a partition satisfying (d) and proceeds to fulfil (c). The Split and Merge scheme starts with a partition which does not have to satisfy either of the conditions (c & d), and proceeds to produce a partition which satisfies both.

It is important for implementing the split and merge algorithm to use a suitable data structure.

8.2 Data Structures

Horowitz & Pavlidis (1976) have proposed a hierarchical data structure called the quadratic picture tree, or quadtree. The quadtree is used for various applications such as image processing, computer graphics, pattern recognition, robotics, and cartography. The most important feature is its hierarchical nature which lends itself to a compact representation. For a good survey of its use see Samet (1984).

The quadtree is an approach to image representation based on the successive subdivision of the image into four equal square subsections. Each of these is divided into four squares recursively until the smallest subsections consist of a single pixel. Each subsection is called a node, the single pixel is called a leaf, and the whole image is associated with the root node. A node is uniquely specified by a given corner, the length of a block side, and the relative position or level in the tree. The quadtree consists of several levels, the leaves are in the lowest level and the root node represent the highest level. Figure 8.1 shows example of a three level quadtree for an image of size 4 x 4.

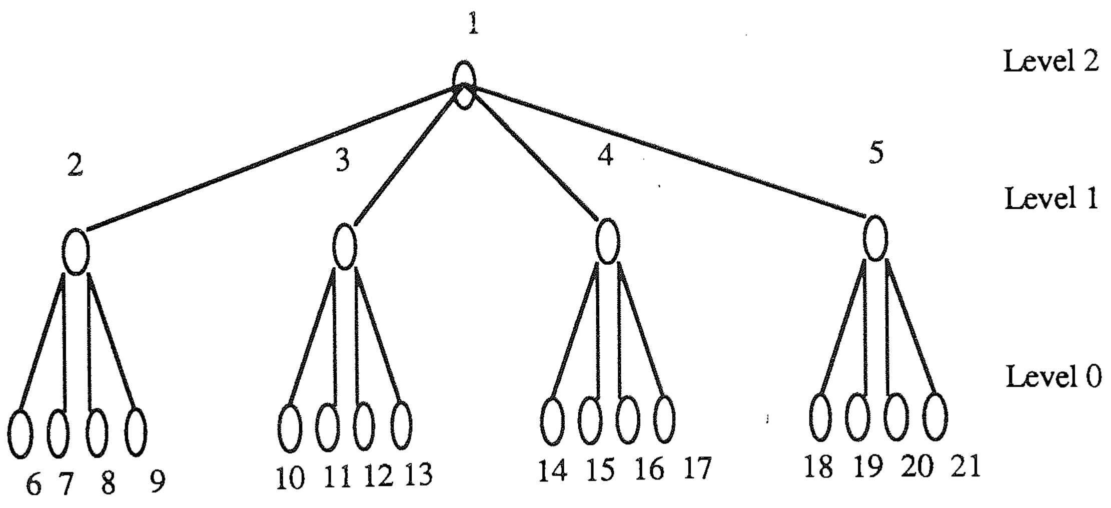


Figure 8.1 Three-level quadtree.

The quadtree can be defined as an undirected graph without circuits since the flow of data can be upward or downward.

For actual representation it is not necessary to use all the nodes of a specific quadtree size. This can be shown by considering figure 8.2.

There is a drawback in quadtree representation shown clearly in figure 8.2: the regions which are adjacent in the actual image are not necessarily adjacent in the quadtree representation. For example, nodes 9 and 10 are adjacent in figure 8.2 (a) but they are not in figure 8.2 (b). To overcome this drawback another type of data structure is used, representing the region adjacency information.

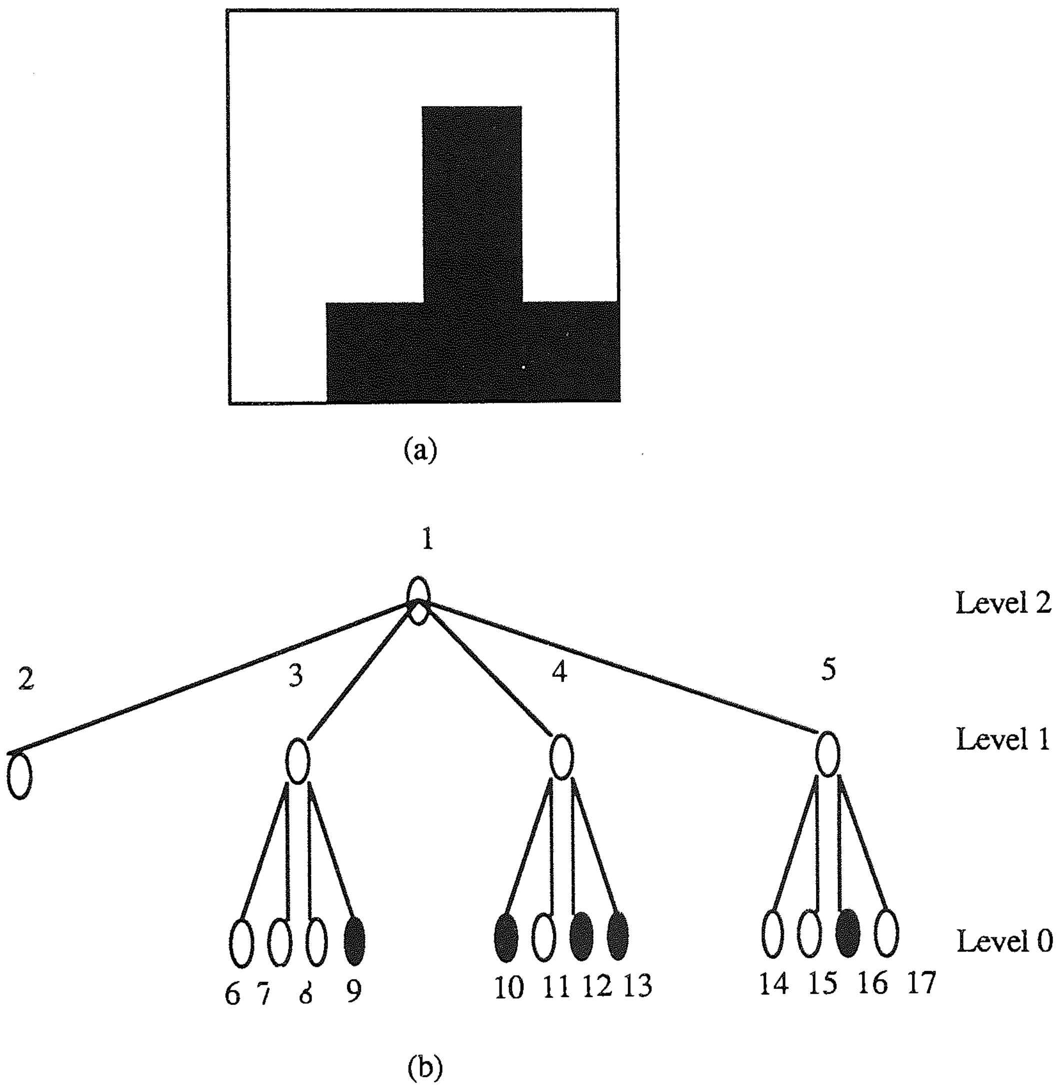


Figure 8.2 The representation of image as a quadtree.

Region Adjacency Graph

Region adjacency graph (RAG) is a graph for describing a segmented image. Each node in the graph corresponds to a region. Branches exist between nodes corresponding to adjacent regions. Figure 8.3 shows an example of RAG of an image.

RAG contains useful information for image segmentation complementary to that held in the quadtree. This is due to the fact that the relation between regions are all included within the RAG but not in the quadtree. However, quadtree is more efficient and fast for initial segmentation. So for efficient and fast implementation of the Split and Merge algorithm the image is first represented by the quadtree for initial segmentation, then the RAG is used to represent this initial segmentation. Region grouping is then applied to complete the segmentation process.

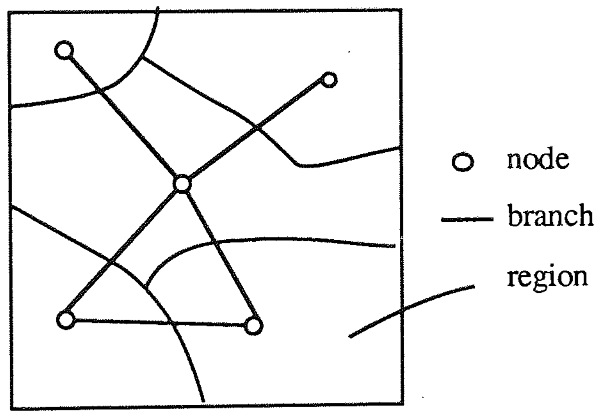


Figure 8.3 Region Adjacency Graph representation.

Steps for S & M algorithm

Step 1: Initialisation

The first step for the algorithm is to build the quadtree for the image on an initial level. This means choosing a certain level in the quadtree. The number of regions for the initial level is equal to the number of nodes in that level since each node represents a region. This is called the initial segmentation or the initial node cut-set.

Each node contains several items of information such as level number, the position of the top left pixel of the region; the length of a side of the region; and the maximum and minimum values of the brightness of the pixels in the region. This information can be represented by a record.

For an image of size (N x N), the block side at level ℓ is $s=2^{\ell}$. For the lowest level, which is level zero, the block side is 1. At the highest level, which is equal to $L=\log_2(N)$, one block represents the whole image. Usually the size of the quadtree is equal to the size of the image.

There are several ways of numbering the nodes of the quadtree which represent the regions for image segmentation purposes.

The numbering system used here is as shown in figure 8.4.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 8.4 Numbering system for three-level quadtree.

For initialisation a certain initial level is chosen and it is called ℓ_o. For this level the size of the block sides is

$$s_o = 2^{\ell_o}$$

the number of nodes n in this level is such that

$$n = 4^{(L-\ell_o)}$$

This is called the initial cut-set which segments the image into square blocks (regions) of equal sizes, the number of which is equal to the number of nodes. The information calculated and stored for each block is as follows:

- Position of the top-left pixel of the block.
- The block size.
- The minimum and maximum of the brightness function f(x, y) for each block.

Step 2: Merge

In this step a comparison is made between each group of four nodes which have a common parent. These nodes can be found by using the following procedure explained below.

Consider the four blocks in figure 8.5.

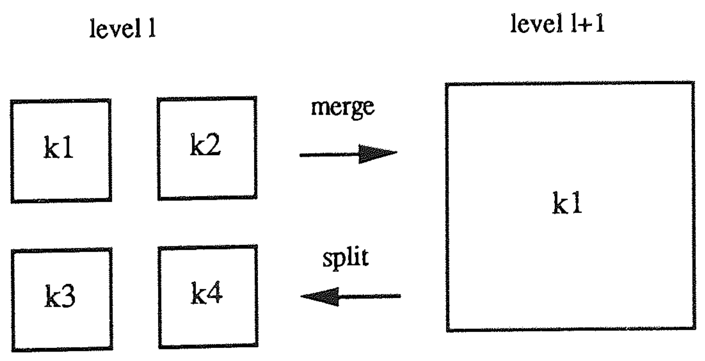


Figure 8.5 The merge and split process.

For a level ℓ such that

$$0 < \ell \leq \ell_o$$

the relationships between the four nodes k1, k2, k3, and k4 are:

$k2 = k1 + 2\,(\ell_o - \ell)$,

$k3 = k1 + 2(2\ell_o - \ell)$,

$k4 = k3 + 2(\ell_o - \ell)$.

The four nodes are merged if their union satisfies the predicate P defined in section 8.1. The merging process is actually the removal of nodes k2, k3, and k4 and then updating k1 with new size and and other relative values. This procedure is repeated at each level from level ℓ_o up to level L-1.

Step 3: Split

This procedure starts after completion of the merge step. Each node which is not merged in the merge step is tested for satisfaction of the predicate P. If it is not satisfied then the block is split into four equal sub-blocks; the process is the reverse of that of the merge one and it is shown in figure 8.5. This procedure is repeated for every set of four nodes until each node satisfies the predicate. Again, this procedure is applied for all the unmerged nodes of the merge step.

Step 4: RAG-merge

In this step a region adjacency graph is built to describe the adjacencies of the blocks. This is then used to merge the adjacent blocks which could not be merged by the quadtree.

8.3 Hierarchical Pyramids

The pyramid architecture has been described in section 3.3.2 and shown again in figure 8.6. It can be seen that the quadtree structure which underlies the split phase of the Split and Merge algorithm can be mapped quite naturally onto a hierarchical pyramid architecture. Therefore the pyramid architecture has been chosen for the parallel implementation of the Split and Merge algorithm (Mansoor & Sokolowska, 1988).

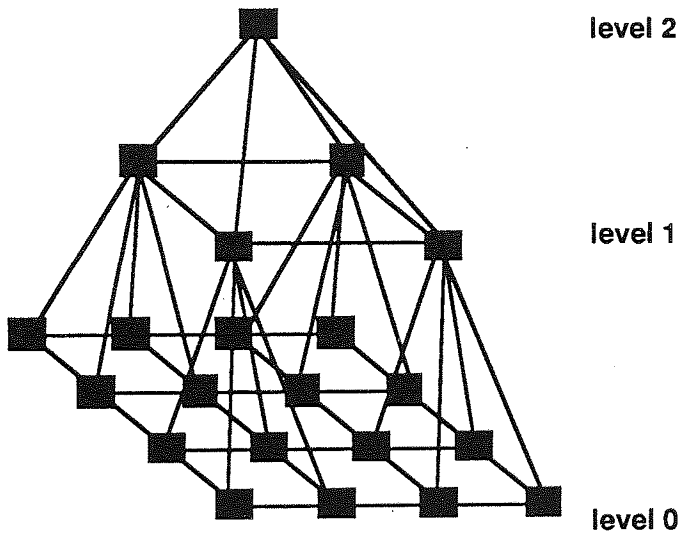


Figure 8.6 Hierarchical pyramid with 3 levels.

The starting step is sending the image to the level 0 of the pyramid in which each Transputer takes one subimage. This step is equivalent to chosing the initial level in the Split and Merge algorithm. Advantage is taken at this stage of the fact that the splitting process in Split and Merge algorithm can be initiated on any level of the image quad-tree and carried out, in parallel, independently for each sub-image. At the completion of the splitting each processing element holds a symbolic description of the local sub-image in the form of a list of blocks, their location, size and grey level properties appropriate for a chosen uniformity criterion. A pictorial map of blocks can either be stored explicitly or be restored from the above description. The merging process which follows does not have to refer any longer to image pixel data - it can rely entirely on the derived symbolic description.

In contrast to the splitting which takes place on the single (lowest) level of the pyramid, the merging uses all the levels of the pyramid, on each level performing the same actions. At first all the uniform regions are merged within one processing element. Then each group of the four children of the same parent exchange data to produce a consistent description on their level and pass the description to their parent, who performs merging, communicates with its three siblings, passes an agreed description to its parent and so on, until the root element of the pyramid is reached and the final segmentation of the entire image is established.

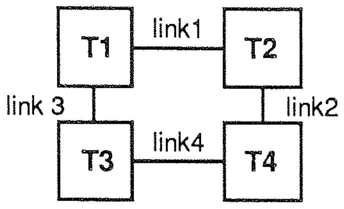
8.4 Processing and Intercommunications

The initial image partition is established in the host Transputer and sub-images sent to appropriate processors at the base level of the pyramid. For the image of dimension N x N and pyramid of size s each processor takes a sub-image of N^2/s pixels. The technique of data distribution was described in section 5.5.

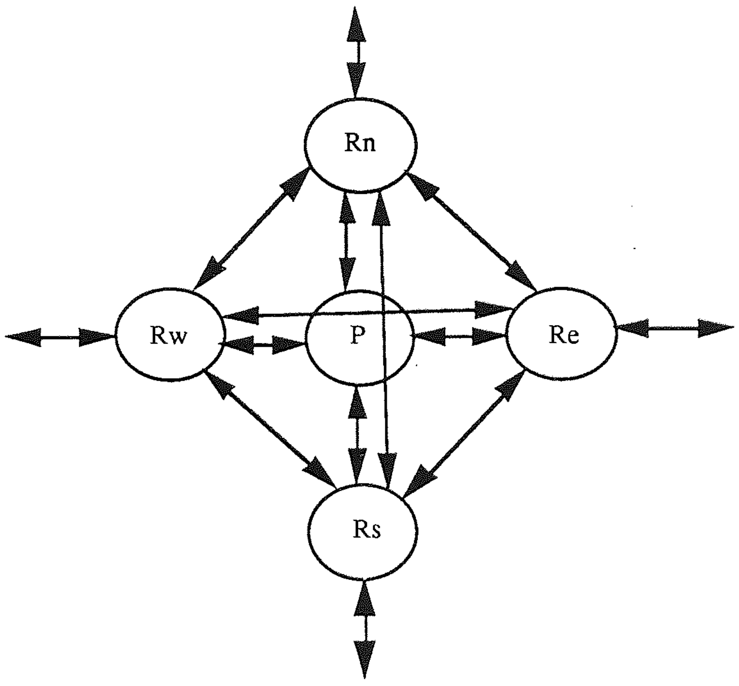
The processors execute the initial part of the Split and Merge algorithm up to the RAG-merge stage; the results are stored in each local memory. Following this each of the four connected processors implements a communication protocol to exchange the local segmentation results. The modified results are passed then to the next level up where a new sub-image of dimension 4 * (N^2/s) is consolidated and subjected to merging. The results are exchanged among each four processors connected at this level using the same protocol. This processing continues until the final result reaches the root processor.

8.4.1 Intercommunication Protocol

The intercommunication protocol is implemented for four Transputers (T1..T4) connected as in Figure 8.7(a).



(a)



(b)

Figure 8.7 S & M: (a) Links between the Transputers for intercommunication protocol. (b)Occam model for each of the Transputers in the network.

The top level program structure of the protocol is as follows;

```
PAR
    P1
    P2
    P3
    P4
```

where P1, P2, P3, and P4 are processes implementing the algorithm running on T1, T2, T3, and T4 respectively. Each process (P1..P4) has the Occam model which was shown in figure 5.6 and redrawn here in figure 8.7(b).

The routing (Rn, Rs, Rw, Re) is the same in all P1..P4. The process P differs in each of P1.. P4.

Each process in figure 8.7(b) contains SEQ construct only. The following describes the parts common to the P process in all four Transputers (T1..T4).

Each Transputer contains the results of the segmentation of a sub-image. These are:

1) A two-dimensional array of the size of the original sub-image array containing for each pixel its region number (aa[i][j]).

2) The number of regions in the sub-image (mm).

3) Two vectors for the maximum and minimum value of the brightness intensity for each region (ls[i],ms[i]).

4) A list of blocks and their properties:

(a) A vector which contains the number of blocks belonging to each region (re[i]).

(b) Linked list for the block numbers for each region, starting with the block numbers of region number one.

(c) Index 'pointer' for the starting position of the first block number belonging to region (i) (kn[i]) in the linked list.

(d) Two vectors containing the x-y position of the top-left pixel for each block (a[i],b[i]).

One in each of the four Transputers is used to control and synchronize the intercommunication between the four Transputers. This is because the intercommunication protocol is of synchronized type, since it contains several sequential phases. T1 may be considered as the controller in figure 8.7(a). Before applying the intercommunication protocol, each Transputer T2, T3, and T4 sends a message to T1, showing that they are ready to apply the intercommunication protocol. These messages are sent after completion of each phase: after the completion of segmentation of their subimages.

The program construct for each Transputers (T1..T4) for implementing the intercommunication protocol is

```
SEQ
        Phase one (modify region number)
        Phase two (merge the uniform regions on the borders
                    between T1&T2 and T3&T4)
        Phase three (merge the uniform regions on the borders between
                    T1&T3 and T2&T4 and merge the regions which are
                    uniform on more than two Transputers)
        Phase four (merge the uniform regions which are uniform one
                    more than two Transputers)
```

The four phases described below. Note that in the following description sending regions means sending the symbolic information describing those regions; these are region numbers, and maximum and minimum values of these regions. The particular cases of the regions crossing processor boundaries are illustrated in figure 8.8 on page 191.

Phase one

In this phase the region numbers are modified to be unique among the four subimages. This is done by communicating the number of regions of the sub-images among the four Transputers shown in figure 8.7. Assuming the sequence is T1 to T4, the region numbers of T1 remain unchanged while for T2 each region number is increased by the number of regions of T1 and so on. This is achieved as follows: T2 receives the number of regions in the subimage of T1 and modifies its region numbers by adding that number to each region number, T3 modifies its region numbers by adding the number of regions in T1 and T2, and T4 modifies its region numbers by adding the number of regions in T1, T2, and T3.

Phase two

This phase examines regions which are neighbours across the Transputer boundaries and merges them if they are uniform. This is achieved by the following scheme:

T2 sends the regions which lie on the border with T1 to T1. T1 compares its regions which are adjacent to those received from T2 and merges similar regions and then sends the new symbolic information of the merged regions to T2. T2 receives those regions and updates them. In addition to that the 'pointer' values are used to distinguish the merged regions from those which have not been changed, this is achieved by putting negative values to them.

The same happens between T3 and T4 , in this case T4 sends the border regions to T3. Here the negative values are given to both regions merged in T3 and T4.

However there are still several special cases which should be considered such as those illustrated in figure 8.8 (b) & (c), or more complicated cases, and these are included in the algorithm and described below.

Phase three

The same procedure is applied as in phase two but the intercommunication processes are between T3 & T1 and T4 & T2 , with the following exceptions:

<u>T3</u>

The values of the 'pointer' for the merged regions are either positive or negative due to the processing operation of phase two. Either of the following actions takes place:

(a) The negative values of 'pointer' mean that the regions concerned have been sent to T4 in phase two (figure 8.8 case f or i or j or l). Therefore in addition to merging these regions, they are marked for later use in phase four; these regions are called the 'stored' regions in T3.

(b) For regions with positive values of the 'pointer', the action is to merge them without changing the sign of their vectors (figure 8.8 case a or g or k).

<u>T4</u>

The 'pointer' values of the merged regions are either positive or negative due to the processing operations in phase two. Either of the following actions takes place:

(a) The regions with negative 'pointer' values have been merged with regions in T3 (figure 8.8 case e or h or i or l). The action is to merge them, store them to be used later in phase 4, and set their 'pointer' values to positive; these are called the 'stored' regions in T4.

(b) For the regions with positive 'pointer' values, the action is to merge them without changing the sign of the 'pointer' (figure 8.8 case a or c or d).

Phase 4

This phase is mainly to check the stored regions in phase three. T1 is not active in this phase and the other Transputers act as follows:

<u>T3</u>

(a) Send the stored regions in phase (3) to T4 (figure 8.8 case f or i or j or l).

(b) Receive regions from T4; modify the regions with negative 'pointer' values (figure 8.8 case e or f).

T4

(a) Receive the stored regions in phase (3) from T3. The regions with positive value of the 'pointer' are those which have been merged with regions in T2 during phase three since they had negative 'pointer' value after phase two. Two actions are applied (figure 8.8 case i or l). First, region number is modified and its 'pointer' value set to negative. Secondly they are sent to T2.

For regions with negative 'pointer' value (figure 8.8 case f or j), region number is modified.

(b) Send a stop message to T2.

(c) Check the sign of the 'pointer' values for regions stored in phase three, send those with positive 'pointer' values (figure 8.8 case e or h) to T3.

T2

The processor T2 is waiting for a region number to come from T4. This applies to regions which belong to T1, T2, T3 and T4 (figure 8.8 case i or l). Only region numbers with a positive 'pointer' value are modified (figure 8.8 i). There is no need to include figure 8.8 case l because regions like that have the correct values from the phase two.

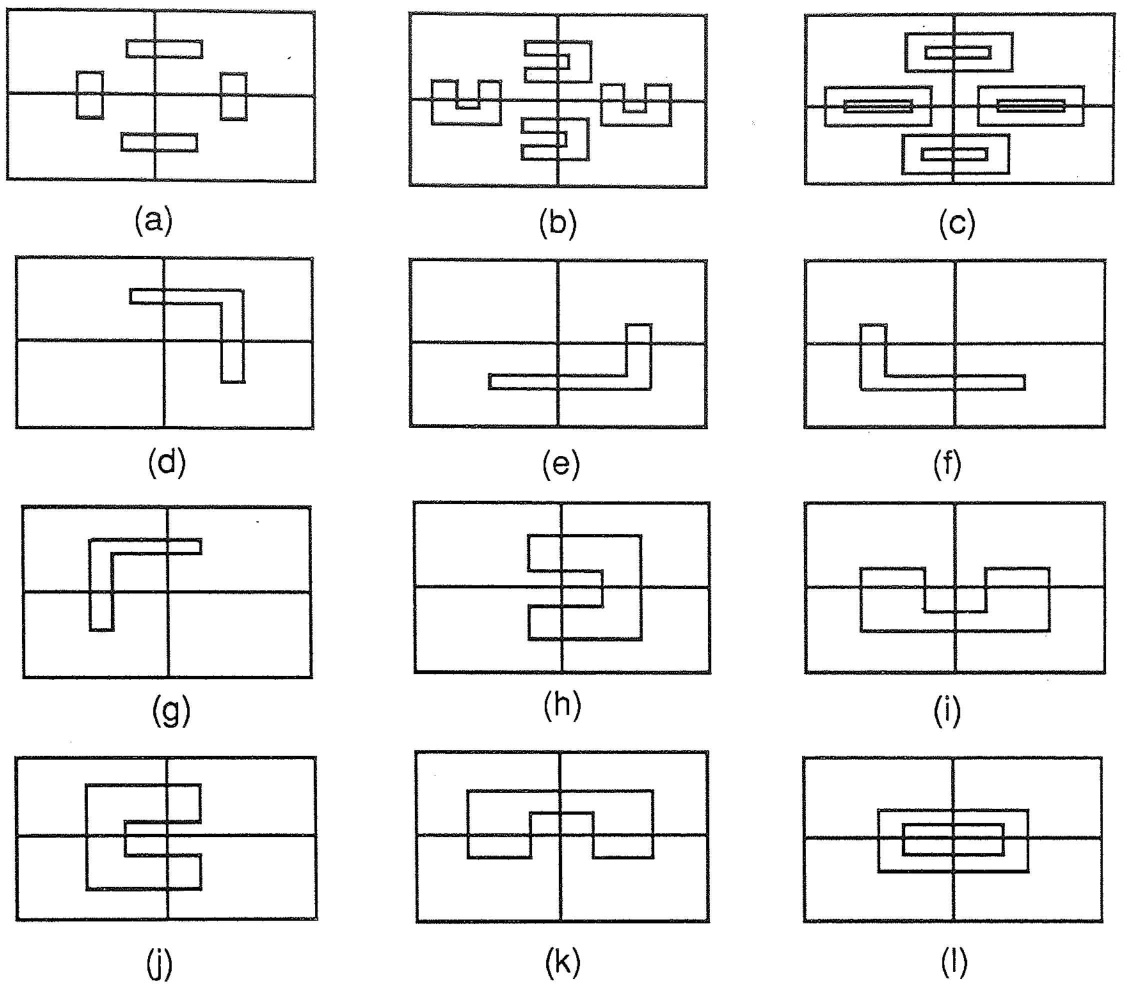


Figure 8.8 S & M: The possible region forms taken in consideration in the protocol.

8.5 System Configuration

The block diagram of the system is shown in figure 8.9. The host Transputer is responsible for accessing the input image and partitioning it into initial blocks. It then sends these partitions to processors at the base level. The segmentation is performed by Transputers arranged as a hierarchical pyramid. The results arrive to the root Transputer which arranges for them to be displayed via G007 graphics board.

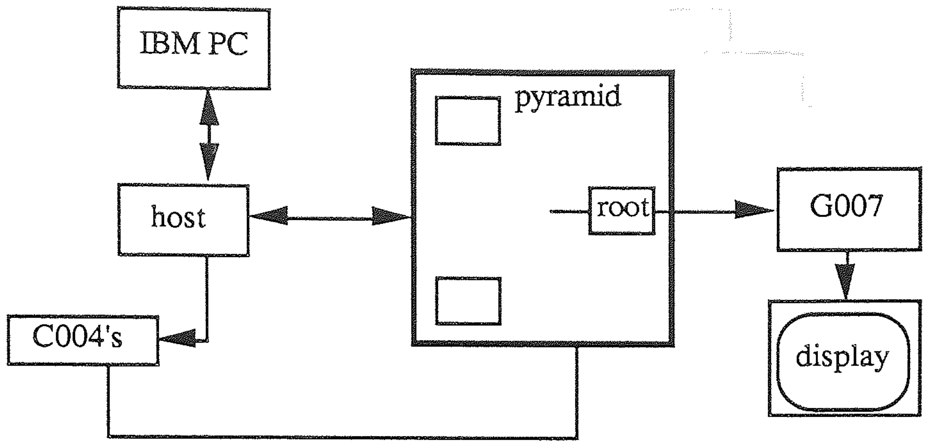


Figure 8.9 S & M: System block diagram implementing the Split and Merge algorithm.

In the figure 8.9 the host and the processing elements in the pyramid are Transputers. In the pyramid, with the exception of the root level, each group of four Transputers is interconnected and each of them is connected to the parent as shown in figure 8.10.

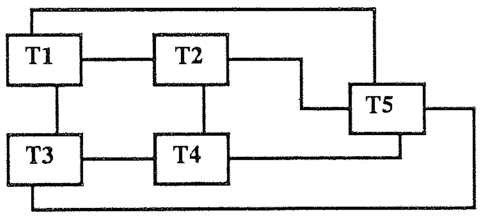


Figure 8.10 S & M: Parent-children communication in the pyramid.

8.5.1 Pyramid of size four System

A program written in Occam has been designed for the implementation of the Split and Merge algorithm (S & M). A pyramid of size four has been built using fixed configuration which is shown in figure 8.11.

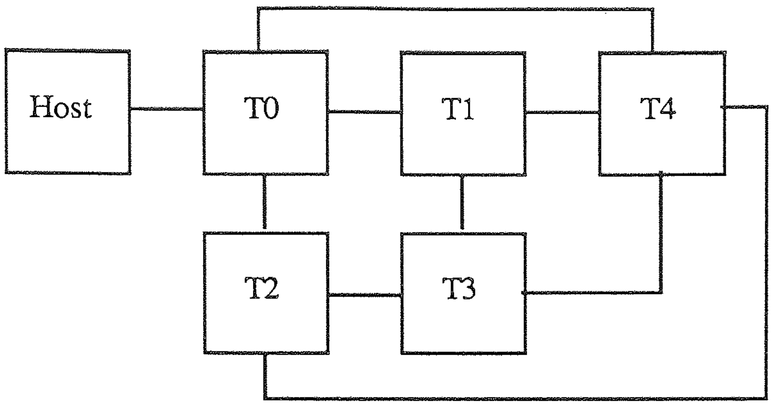


Figure 8.11 S & M: Pyramid of size four system.

The implementation of the S & M algorithm for pyramid of size four is in fact the implementation on mesh of size four. This is because the root Transputer in the pyramid is just collecting the result. For this reason the host in figure 8.11 is used to collect the result and T4 is excluded in the implementation.

A timer of Transputer T0 is used to measure the time spent for the initial S & M algorithm and that spent for intercommunication protocol Tc. The total time is measured using the timer of the host Transputer.

For the purpose of comparing the performance of the system with different types of image, four images with different number of regions are used. The first image is a plain image which contains one region, the others are checker boards with 8x8, 16x16, and 32x32 blocks. These are chosen to observe the effect of the communication overhead (Coh), which is the ratio between the time of spent for the intercommunication protocol to the time spent for the initial Split and Merge algorithm.

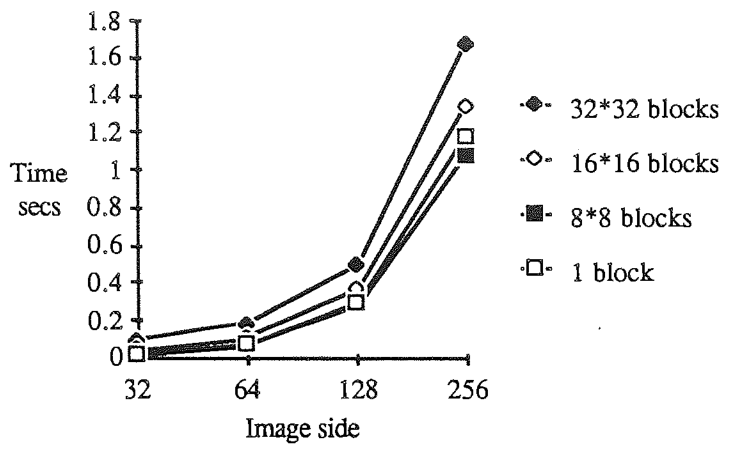
8.5.2 Performance for Pyramid of size four

In order to see the effect of changing the communication overhead (Coh) on the performance of the system two initial level values are used for the Split and Merge algorithm. This is important especially to observe its effect on the same image.

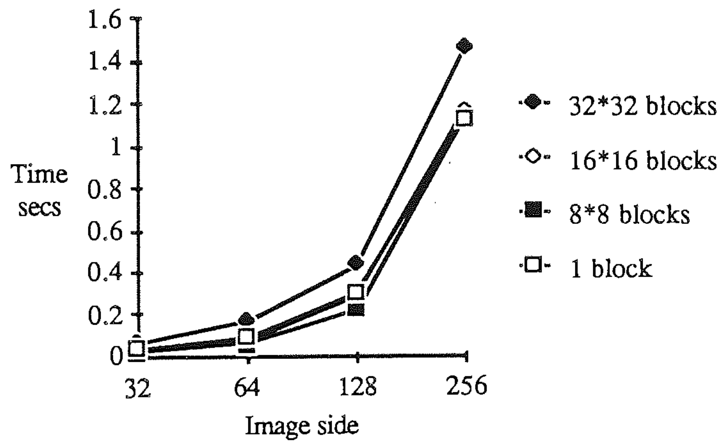
The measurement of the time is split into two types. The first is the time of the actual processing (actual time) of the Split and Merge algorithm, which is sum of the time spent for the initial S & M processing plus that spent for the intercommunication protocol processing. The second is the total time, which includes in addition to the above, the time spent for data distribution and collection. The discussion of each of them separately is important since it will show the effect of the data distribution on the overall performance.

The actual time is discussed first. Figure 8.12 on the next page shows the time diagram with respect to the image size and for the four image types.

The image which contains more blocks takes longer time to be processed since it takes longer time during the initial partitioning and the intercommunication processes because there are more blocks of information to be interchanged.



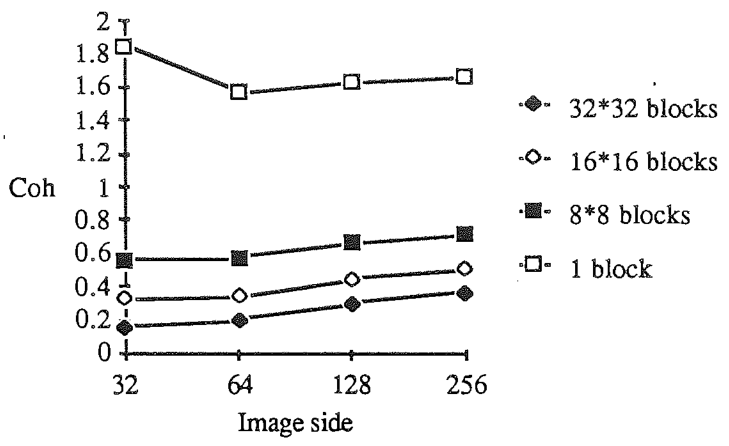
(a) lo=1



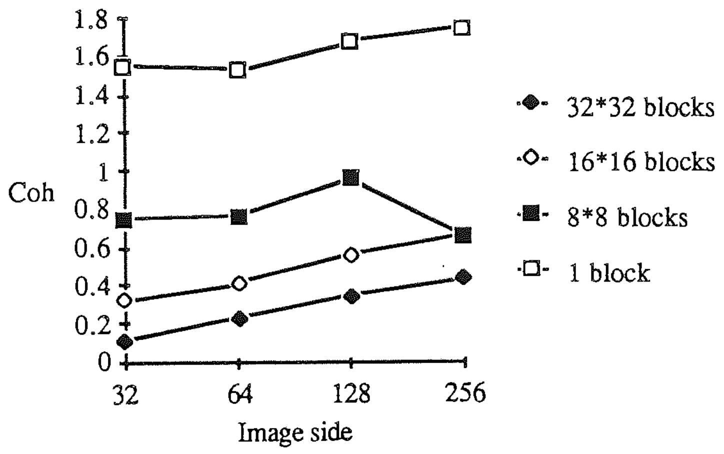
(b) lo=2

Figure 8.12 Actual time of S & M algorithm on four Transputers.

Figure 8.13 shows the Coh values with respect to the image size.



(a) lo=1

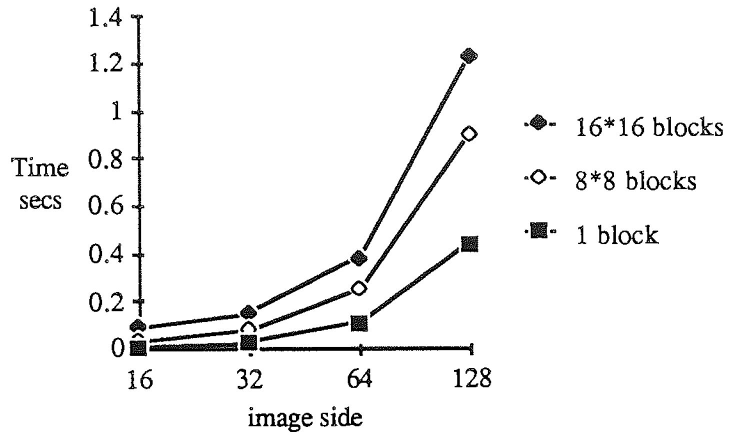


(b) lo=2

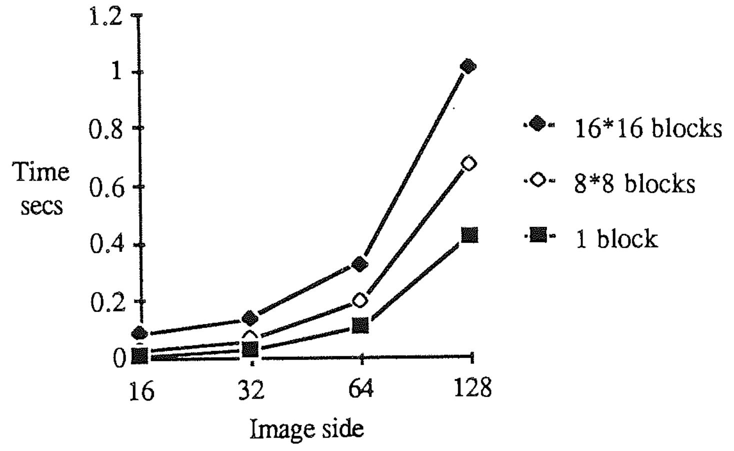
Figure 8.13 S & M: Coh for the actual time on four Transputers.

The Coh for the plain image is higher than for the others during the intercommunication protocol, in spite of the fact that there is only one region to be sent between the Transputers. This is because a relatively long time is spent on updating the whole subimage as it is being merged with other subimages. For other images Coh is higher for those with less blocks since the time of processing to get the initial partitioning is higher for those with higher number of blocks. The Coh for these is higher when initial level is two than when it is one. The reason for that is that for the initial partitioning processing the time is lower for higher initial level.

For calculation of speed-up and efficiency, the time of processing for one Transputer is measured and plotted in figure 8.14.



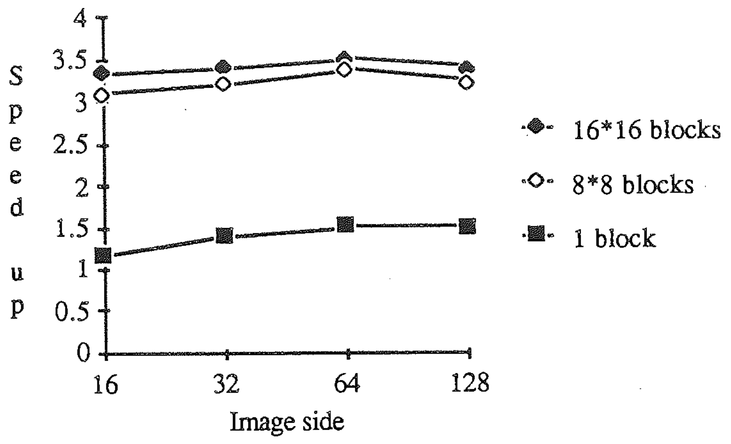
(a) lo=1



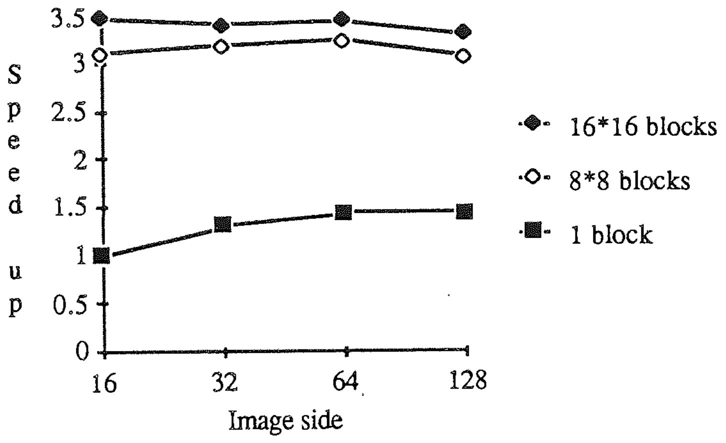
(b) lo=2

Figure 8.14 S & M: Actual time of processing for one Transputer.

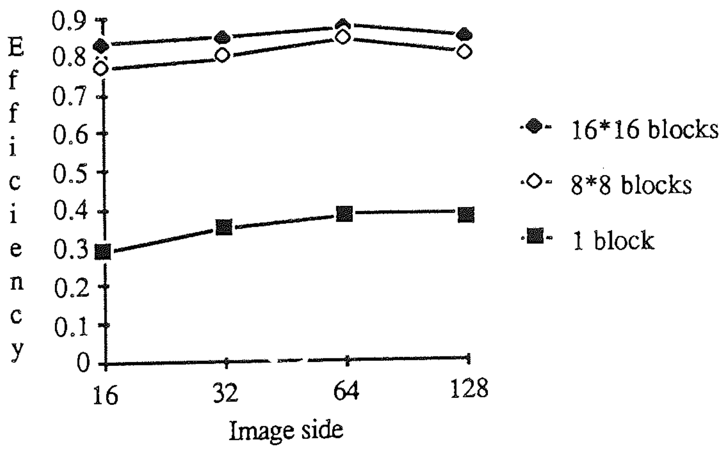
The speed-up and efficiency graphs for four Transputers are shown in figure 8.15.



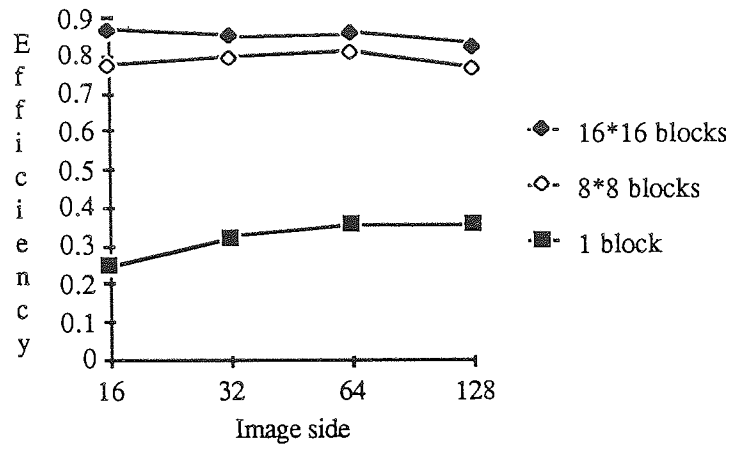
(a) Speed up with lo=1



(b) Speed up with lo=2



(c) Efficiency with lo=1

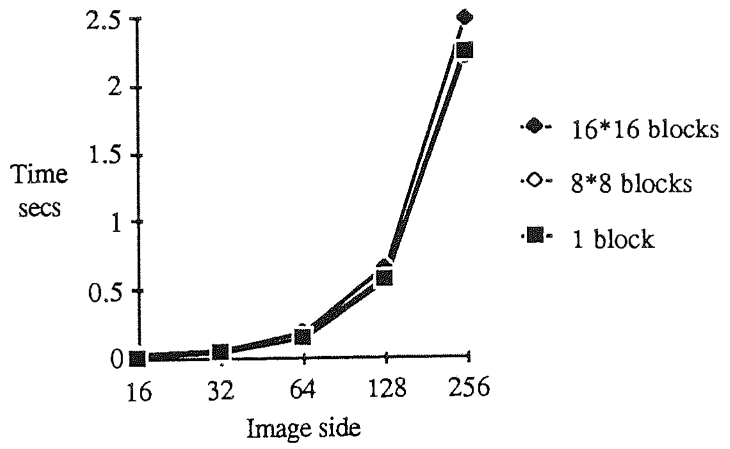


(d) Efficiency with lo=2

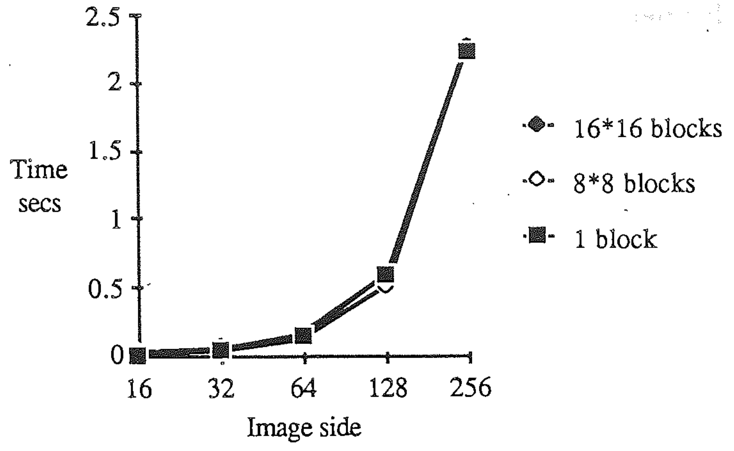
Figure 8.15 S & M: Speed up and efficiency for four Transputers for actual time.

The speed-up is inversely related to the value of Coh. This is due to the fact that the time spent for intercommunication between Transputers is higher than that spent inside the Transputer, as in the case of one Transputer. The efficiency is related to the speed up, so that the better efficiency is achieved for better speed up.

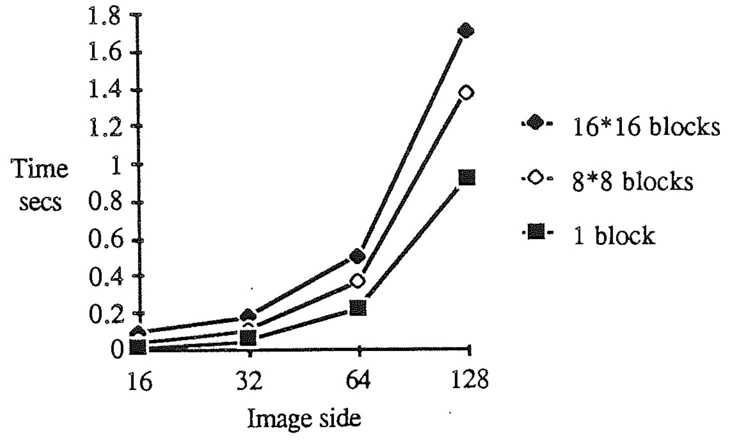
The total time is illustrated in figure 8.16 which shows the time graph with respect to image size for the four images.



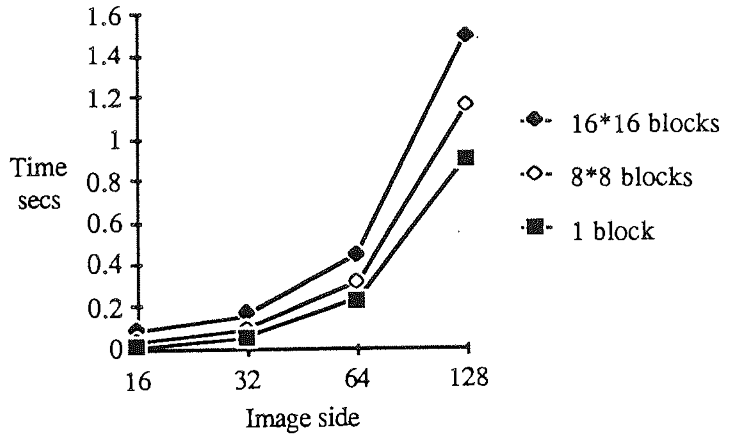
(a) one Transputer total time with lo=1



(b) One Transputer total time with lo=2



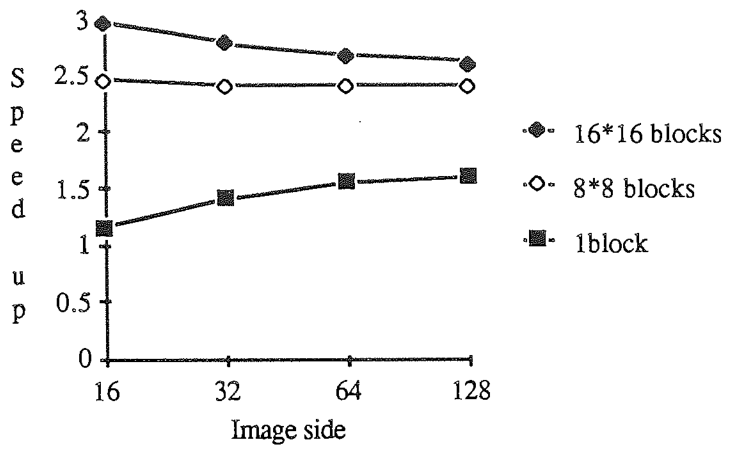
(c) Four Transputer total time with lo=1



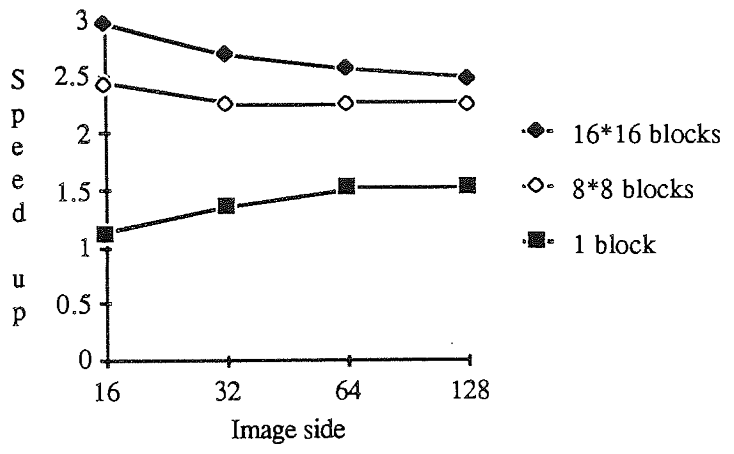
(d) Four Transputers total time with lo=2

Figure 8.16 S & M: Total time for one and four Transputers.

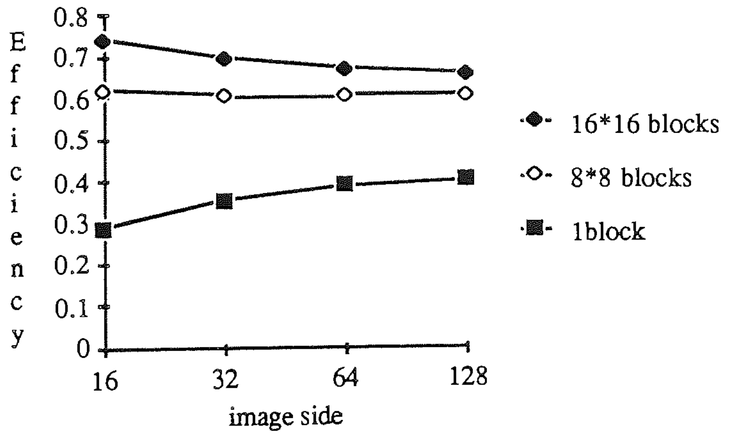
The speed up and the efficiency graphs for the total time are plotted in figure 8.17.



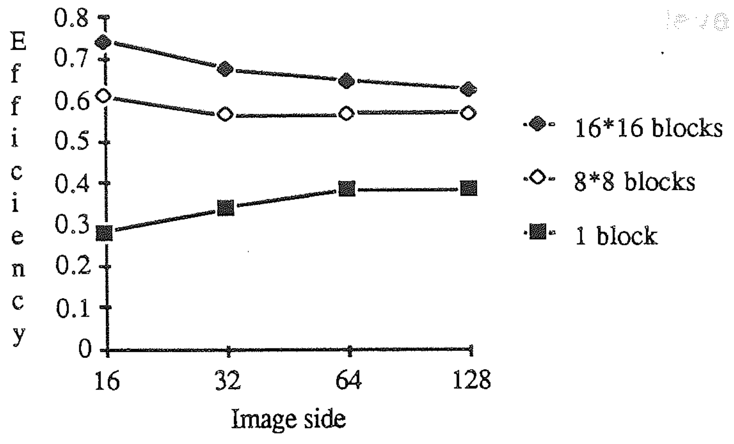
(a) Speed up with lo=1



(b) Speed up with lo=2



(c) Efficiency with lo=1



(d)Efficiency with lo=2

Figure 8.17 S & M: Speed up and efficiency of the system.

It is clear from graphs in figure 8.17 that the speed-up decreases in the case of total time measurements since the ratio of the time spent for data distribution and collection is high with respect to the actual time of processing of the Split and Merge algorithm. It can be concluded that better performance can be achieved if the technology achieves faster intercommunication speed.

It can be seen from figures 8.15 and 8.17, that the relationship between the speed up and the image size is fairly constant for each type of image. This means that for larger image sizes, the speed-up is likely to be similar to that for an image of size 128x128.

8.5.3 Pyramid of size sixteen

Ideally, processor links between the levels for pyramid of three levels should be arranged as in figure 8.18 below, with similar arrangement for pyramids of more than three levels.

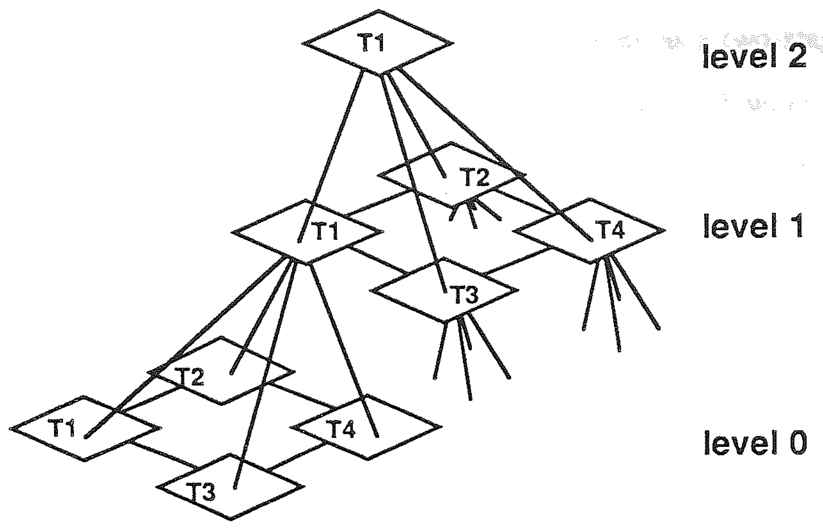


Figure 8.18 S & M: Links between the levels of the pyramid.

It can be seen, however, that each processing element on the intermediate level (level 1 in the figure) needs seven links while the Transputer provides only four. This problem can be solved by using a reconfiguration switch like C004 (INMOS, 1989) which is capable of changing switch settings dynamically between 32 links.

Two different configurations are needed for dynamic implementation of pyramid architecture using Transputers with four links. In the initial configuration, further referred to as the *steady state*, each level is connected to the other levels for data transmission between levels. For full support of the pyramid architecture in this configuration there should be five links for each Transputer, one to be connected to the parent and four to the sons. This problem can be solved for four-link Transputer by using a longer path for transmission. The second configuration is that which supports the implementation of the intercommunication protocol within the levels. This is can be achieved by sending a control message from each level to the controller which is responsible for reconfiguration saying that it is ready to transmit and receive data. The controller reconfigures the links to the second configuration after receiving that control message from all levels.

The processing for a pyramid of size sixteen can be seen as a two-stage pipeline. The first stage is at level zero (base level) of the pyramid where each four Transputers apply the initial S & M algorithm to their subimages, apply the intercommunication protocol, and pass the results to the upper level.

The second stage is at level one where there are four Transputers receiving data from level zero, applying the intercommunication protocol only, and passing the results to the root which is responsible of collecting the final result.

In practice, to achieve the pyramid of size 16, the network needs to be configured in two different ways for better throughput. These configurations are shown in figure 8.19 (a)-(b) on pages 205-206.

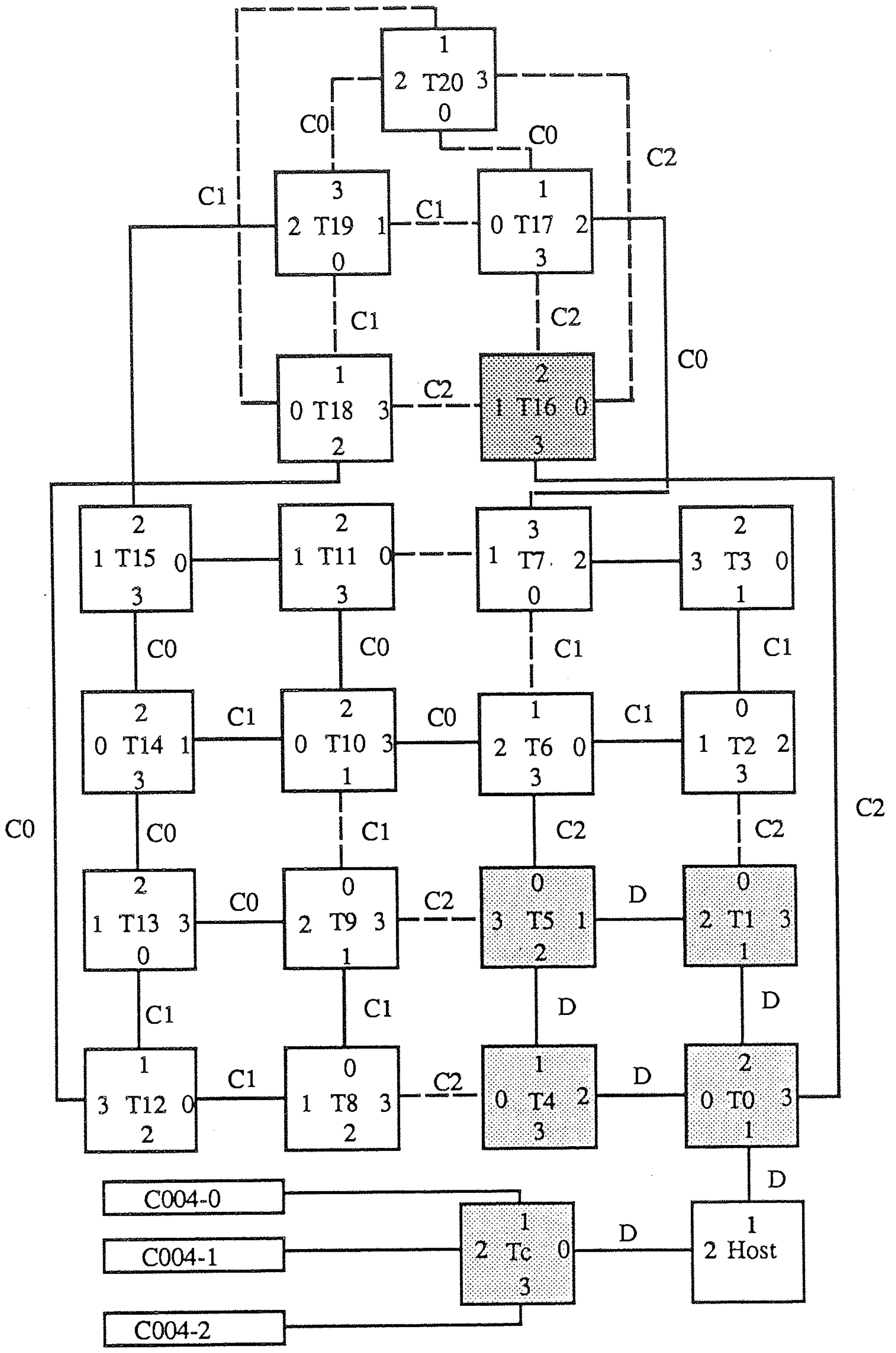
Tc is the controller which is responsible of reconfiguring the links via the three link switches, C004-0, C004-1, and C004-2. Transputers T0 to T15 belong to level 0. Transputers T16 to T19 belong to level one. T20 is the root Transputer of the pyramid.

The procedure of processing is as follows, assuming that there is a sequence of input images:

1) During configuration one (C1), the Transputers in level zero receive the data from the host Transputer, apply the processing of this level to image number (k+1), and send a control message to the control Transputer Tc, which is responsible of reconfiguring the architecture via the three C004s.

The Transputers of level one apply the intercommunication protocol to image k, send the result to level two, and send control message to Tc.

2) During configuration two (C2), Transputers of level zero send data to those of level one.



---- This connection will be changed in configuration 2

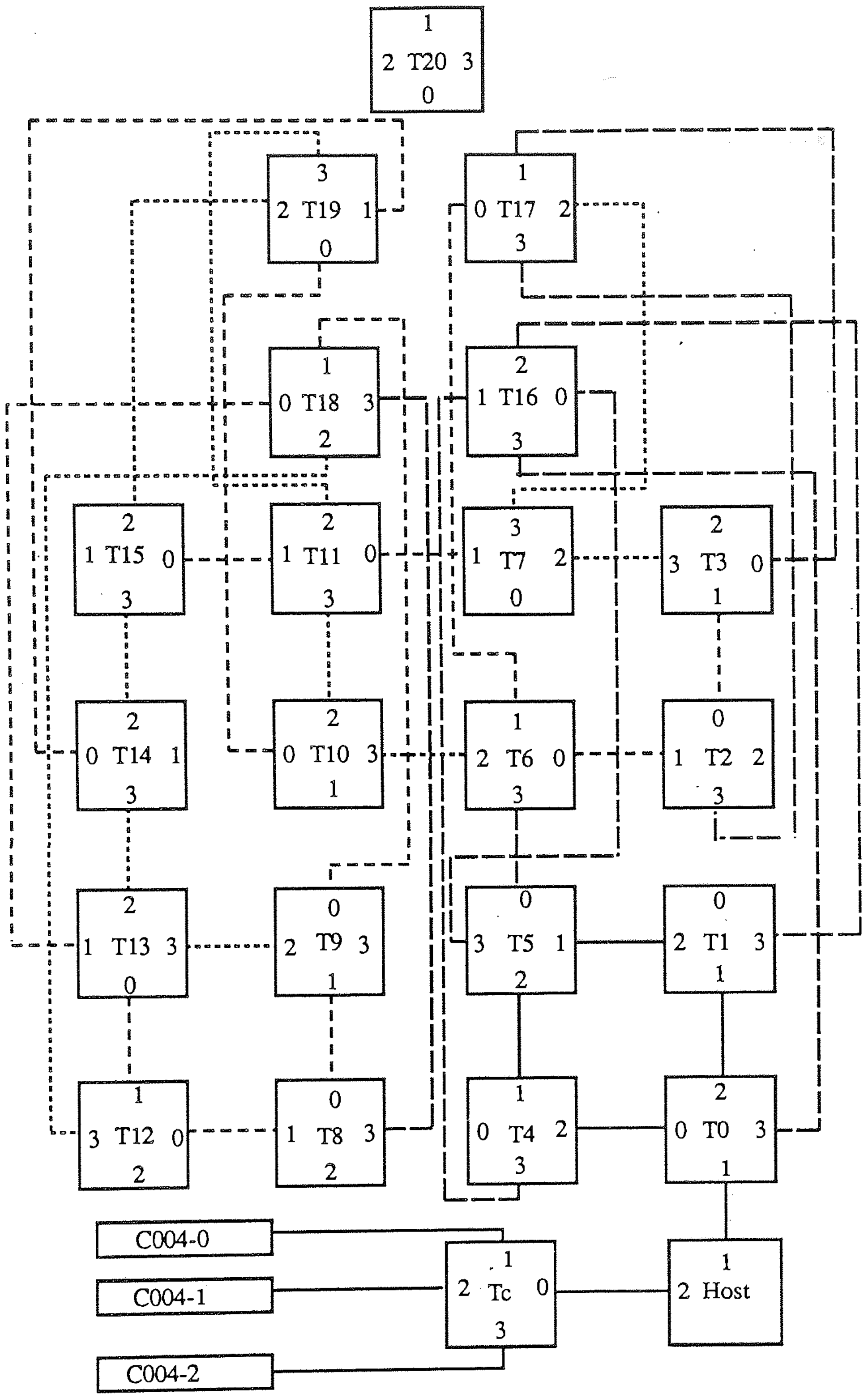
—— This connection will not be changed in configuration 2

C0, or C1, or C2 show that the connection has been implemented via C004-0, or C004-1, or C004-2 respectively.

D is for direct connection. Tc is the control Transputer.

░ Transputer of B008 board ▒ Transputer of G007 board

Figure 8.19 (a) Pyramid of size sixteen; configuration one.



┈┈┈┈┈ This connection is implemented via C004-0

- - - - - This connection is implemented via C004-1

— — — This connection is implemented via C004-2

———— This is direct connection

Figure 8.19 (b) Pyramid of size sixteen; configurationt wo.

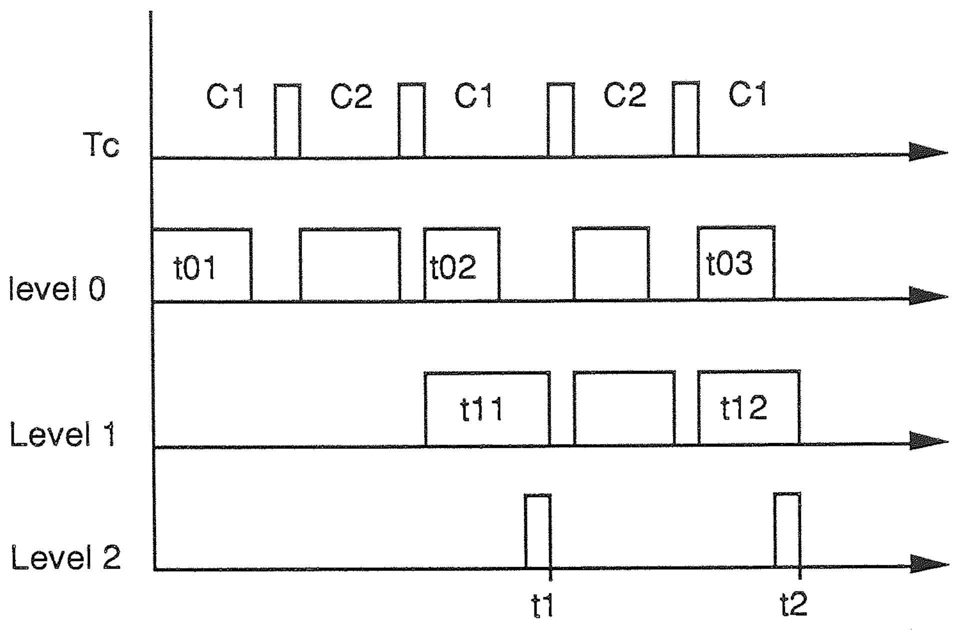


Figure 8.20 S & M: Timing diagram for pyramid of size sixteen. tmk is the time of level m processing of image k. tk is the time at which image k is processed.

Figure 8.20 shows the timing diagram for this procedure.

The above procedure may not represent the optimal solution from a load balancing point of view. This can be found out after its implementation in real time. The reason for not implementing it is because of an insufficient number of Transputers available in the department. However, the estimation of the performance of the S & M algorithm on pyramid of size sixteen has been achieved. This is based on the real time implementation of an array of sixteen Transputers of zero level processing and an array of four Transputers for implementing level one processing. Figure 8.21 overleaf shows the ratio between the processing time T1 of level 1 to the processing of level zero T0 with respect to image size. This is plotted to see the load balance in the implementation.

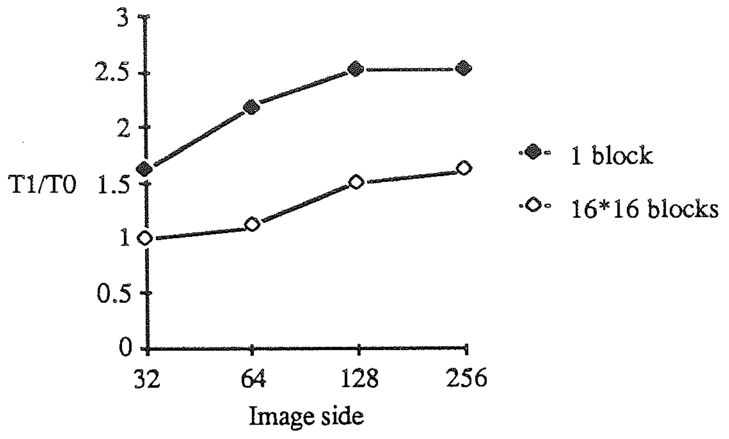
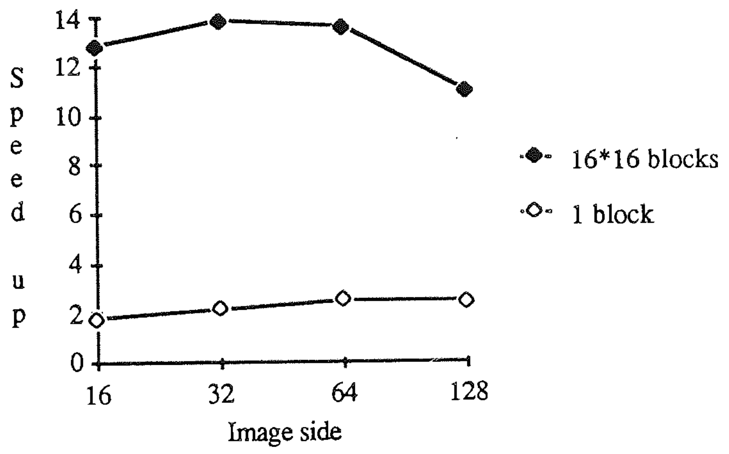
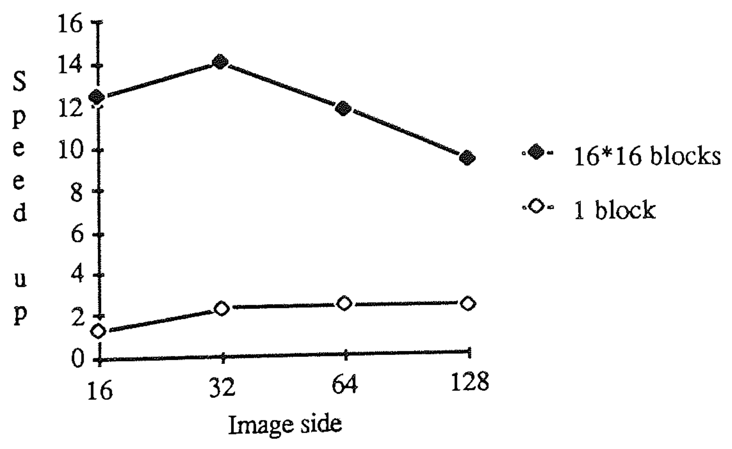


Figure 8.21 S & M: Load balance in pyramid of size sixteen.

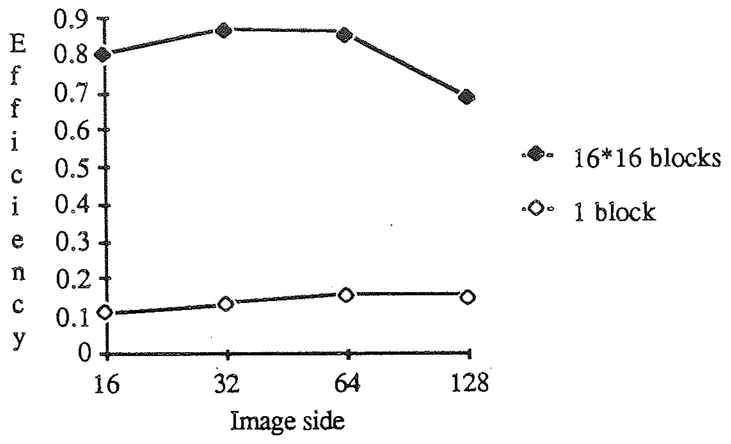
The time measurement for this estimation is based on actual time of processing excluding the time for data distribution and collection. The performance of the system is calculated using the actual time of single Transputer shown in figure 8.14. The speed up and the efficiency graphs are plotted in figure 8.22.



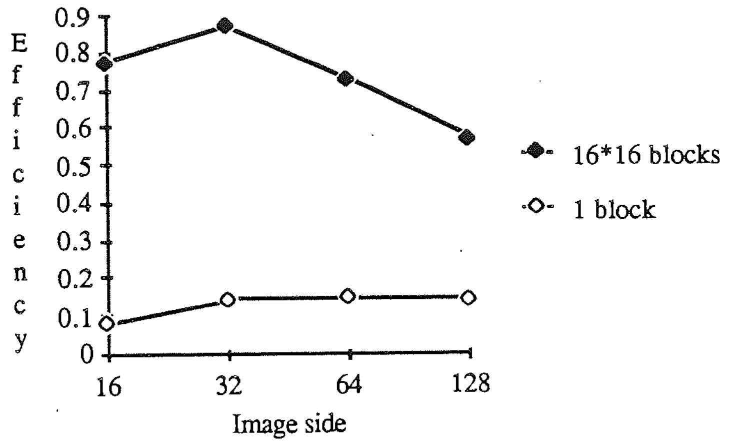
(a)Speed up with lo=1



(b)Speed up with lo=2



(c) Efficiency with lo=1



(d) Efficiency with lo=2

Figure 8.22 S & M: Speed up and efficiency for pyramid of size sixteen.

The graphs confirm that in the case of high intercommunication overhead, as in the case of plain image which represents the worst case, the performance is poor. In other, more realistic cases with low Coh, the performance is good. The performance for the image with more blocks decreases for larger image sizes because there is more data to be intercommunicated within the same level and between the levels.

The results obtained from the analysis of implementation of the algorithm on pyramids of size four and sixteen show that the performance of the proposed parallel S & M algorithm depends great deal on the type of images under consideration.

8.6 Summary

The Split and Merge algorithm for image segmentation, described in this chapter, uses the quad-tree as its main data structure. This hierarchical structure was found to correspond very well to a pyramid architecture. A concurrent version of the algorithm has been designed and implemented on a pyramid of Transputers. The performance of this new implementation has been found to depend on the type of the image being processed. A typical speed-up achieved is about 3.5 but falls down to 1 for the worst case. As the speed-up is inversely related to the communication overhead, better performance may be achieved by using very high speed communication links. This suggestion may be implemented by, for example, fabricating a single chip hosting a number of processors, but this requires further advances in the technology.

Chapter 9

Summary and Discussion

This thesis was concerned with the investigation of practical issues related to the implementation of several classes of image segmentation algorithms on parallel architectures. The Transputer was used as the basic building block of hardware architecture and Occam was used as the programming language. Comments and conclusions related to the implementation of particular algorithms were provided in the concluding sections of relevant chapters. This chapter summarises the main implementation work completed in the course of this project; it comments on general issues related to the use of the Transputer system as a development tool for image processing applications; and on the issues related to the engineering of concurrent image processing applications.

9.1 Summary

Image segmentation

Image segmentation has been identified as one of the bottle-necks of computer vision processing. This is because the large volume of image data is involved in segmentation processing and a number of different features may be required to be extracted from the image data. The segmentation methods chosen for implementation were the convolution, for edge-based segmentation; the Split and Merge algorithm for segmenting of non-textured regions; and the Granlund method for segmentation of textured images. The algorithms chosen were examples of commonly used image segmentation methods, representative of the three main approaches to segmentation, and represented different computational schemes.

Convolution

Three different convolution methods were implemented. This effort was justified by the fact that each of the methods showed different advantages in different algorithmic contexts. The direct method is based on convolution in the spatial domain and was implemented on the array of Transputers. The other two methods are based on convolution in frequency domain. They both require that the Fourier transform of an image is obtained first, followed by multiplication of frequency representations of the image and the kernel, followed by the inverse Fourier transform of the convolved image. Thus the Fourier transform plays a crucial role in implementation of convolution.

Two-dimensional Fast Fourier Transform

Two methods were implemented for the parallel implementation of the two-dimensional Fast Fourier Transform (2D FFT); the Row-Column (RC) and the Vector-Radix (VR) methods. The array architecture was built for the first one and pyramid architecture was built for the second one. The parallel implementation of the 2D FFT RC method includes the serial algorithm for the 1D FFT and the parallel implementation of a matrix transposition algorithm. A novel design was proposed for the matrix transposition. The algorithm was implemented on arrays of Transputers of size four and sixteen; however it is a general algorithm which can run on Transputer arrays of size 2^L, for any value of L.

The pyramid architecture was chosen for the 2D FFT VR as it suited well the structure of the algorithm. The 2D FFT VR implementation was an example of the use of algorithmic parallelism as it relied on the distribution of the consecutive stages of the algorithm among the levels of the pyramid. There were two schemes adopted for its implementation: the top-down scheme based on the decimation-in-time strategy; and the bottom-up scheme based on the decimation-in-frequency strategy. Examples of pyramids of size four and sixteen were implemented. For larger pyramid sizes careful attention needed to be given to the distribution of the algorithm stages in order to achieve good load balance.

The pyramid seems to be a versatile general architecture suitable for implementation of one and two dimensional Fast Fourier transforms based on different algorithmic approaches. For example, the implementation described above can be easily modified to perform the Radix-4 1D FFT because the structure of that algorithm is based, too, on the 'butterfly' operation for four inputs and four outputs.

The overall speed of the execution was higher for 2D FFT VR method than for the 2D FFT RC method; the reason for this lies in the structure of the algorithms themselves, as explained in Chapter 6. The speed-up and the efficiency measures, which relate the performance of an algorithm on a network to its performance on a single Transputer, were similar for both methods, and both showed values close to their respective theoretical maxima.

The convolution through the 2D FFT RC method uses data parallelism so that the same sequence of operations (forward FFT - multiplication - inverse FFT) is applied to all the sub-images in parallel. The convolution through the 2D FFT VR method uses algorithmic parallelism. It was found that for this method the best data flow can be achieved by using for the first FFT the top-down scheme, based on the decimation in time strategy, and for the second FFT the bottom-up scheme, based on the decimation in frequency strategy (see figure 6.26). The performance measures for the FFT based methods related in straightforward way to the performance measures of their respective FFT transforms. The direct method showed a good performance for convolution with small kernel sizes. For larger kernel sizes the FFT technique was better since it required smaller number of operations. Overall, the 2D FFT VR method was deemed to be the best among those implemented, for the reasons discussed in section 6.8.

Split and Merge

A concurrent version of the Split and Merge (S & M) algorithm was implemented on a pyramid architecture. This architecture was selected because it provides the best match for the quadtree, which is the natural data structure for this algorithm. This new algorithm uses both the spatial and the algorithmic parallelism. The first two stages of the S&M algorithm (the initial merge and the split stages) are executed at the base of the pyramid where the unaltered algorithm is applied in parallel to image data in sub-windows. The final stage of the S&M (the RAG-merge stage) is executed at the remaining levels of the pyramid where the homogeneous regions are merged between any four Transputers using a sophisticated intercommunication protocol. This protocol represents a novel solution for merging. A pyramid with more than two levels (L) can be used as a L-stage processing pipeline where a sequence of images is passed from the base level upwards. An important feature of this implementation is its generality since it may be used unchanged for different pyramid sizes and different image sizes.

The analysis of parallel implementations on pyramids of size four and sixteen showed that the performance of the developed algorithm depends very much on homogeneity of images under consideration. Better performance could be achieved by using very high speed communication links. Such links already exist in the new version of the Transputer. Even higher performance should be achieved when all the processors for a certain pyramid size can be fabricated on a single chip; this solution, however, would require further technological advance.

9.2 Transputer system for image processing applications

Transputer hardware

Transputer has a number of features, important for building parallel systems, which other off-the-shelf microprocessors were lacking at the time. Probably the most important feature is the presence of four very high speed serial links through which the Transputer can be connected to other Transputers. The availability of these links overcomes the main drawback of using a common bus, namely the traffic congestion which decreases the intercommunication speed. The availability of the DMA circuit in the Transputer further speeds up the intercommunication process. Transputers in a network can be connected either directly or by using the IMS C004 link switch, so that the configuration can be either static or dynamic as in the case of the reconfigurable system designed for implementing a pyramid of size sixteen, described in Chapter eight, section 8.5.3.

One of the limiting factors in the use of Transputers for image processing applications is the small size of the on-chip memory. At present only 4 Kbytes are available on the chip whereas the minimum memory size required for image processing system is in the range of hundreds of Kbytes. The technology is not yet capable of fabricating such a large memory on a chip and the most feasible solution at present is to speed up access to an external memory and to improve the intercommunication speed. The optical communication using the fibre optics could provide such a solution.

Occam

An important factor from the point of view of the system developer was the availability of a high-level programming language, Occam, based on the communication of sequential processes (CSP) theory and designed specifically for the Transputer and Transputer network programming. The principal new feature, which directly supports parallelism, is the PAR construct. Its use must be carefully considered, to avoid program deadlocks. Programming in Occam does not normally pose many difficulties for someone familiar with an imperative language like Fortran or C, because typical high-level language features are supported.

This language, well suited for implementing concepts of parallelism and communications, was found to have some limitations when applied to specific image processing problems. One of such major limitations is the lack of support for *variable* length *multidimensional* arrays within the channel message protocol. Ideally, the size of an array passed within a message should be passed within that message. This is possible for (one-dimensional) vectors, where the size of a vector may be sent within the message of the protocol together with the vector itself, and can be easily adapted for the use for fixed-size image arrays. For the applications requiring variable image size, as in this project, it was a major limitation which has been overcome by using a vector for image distribution. This, however, has necessitated the use of an additional procedure for conversion between the vector form and the two-dimensional array form. As a result, it increased total processing time and required extra memory for the vector.

Another difficulty arises from the fact that Occam does not allow the user to share variables between the processes running in parallel. This leads to an increase in memory requirements for a program. This limitation affects especially image processing applications because variables representing large two-dimensional image arrays have to be duplicated within each process. One of the consequences of this limitation was that the image sizes used in this project had to be restricted to 256x256.

A limitation of a general nature is that Occam does not support recursion, which would had been an appropriate implementation technique for a number of algorithms such as Split and Merge segmentation or 2D FFT VR method. This limitation was overcome in this work by the use of the WHILE construct in place of recursion.

Development tools

The Transputer Development System (TDS) supports many tools which were found to be very useful in the process of development. The mathematical and the file server libraries in the latest version of the TDS (IMS D700D) were very useful. Since this project started with the first version of the TDS, for which neither mathematical functions nor file server libraries were available, they have been written as a part of the project and kept in user libraries. This ability to create user libraries helped also to maintain uniformity among the programs through inclusion in these libraries the common constants, procedures, and separately compiled utility programs. The debugging tool provided within the TDS allowed for speedy tracing and correction of errors in programs under development. A serious limitation in the usefulness of this tool is that it is not possible to inspect with it the state of logical channels. That made the task of inspection of the channels very difficult and had to be overcome by using control statements within the program. Another frustrating thing in the development system was the very long time it took to compile a program; for example compilation and code extraction for a program with five separate compilations modules could take up to half an hour.

9.3 Other conclusions

This work has prompted several conclusions related to engineering of concurrent image processing applications in general.

The use of data parallelism should always be considered for parallel implementations of image processing applications, especially those belonging to the low-level vision class. This is because of the large amount of data involved in the processing and (normally) highly homogeneous processing. Algorithmic parallelism should be considered for the implementation of intermediate-level algorithms as processing becomes more data-dependent and algorithms become more complicated. However, on the intermediate level processing the amount of pixel data can still be very large and there is normally scope for combining algorithmic parallelism with data parallelism. This applies especially to algorithms which contain several stages of processing such as, for example, the Fast Fourier Transform and the Split and Merge algorithm. Such combined schemes were presented for the above algorithms in Chapter six and Chapter eight, respectively.

As there exist an almost infinite number of different computational schemes for image processing, it is impossible to have a single parallel architecture which is equally suitable for all the algorithms. One possible solution to this problem is to use a reconfigurable system. This solution is particularly suitable during the development stage because alternative architectures can be explored. The Transputer system, including the software switcher, proved to be a very good tool for this type of work. Its success in this field can be attributed to the following features.

The Transputer's inbuilt links, through which it can be connected to other Transputers, make it possible to build large networks with different topologies. It should be noted that it is possible to implement topologies which require more than four links provided directly by the Transputer.

The connections between Transputers can be fixed or they can be configured dynamically by software using an automatic switch such as the COO4. This latter feature makes it possible to alter the configuration at run-time and thus to provide a most suitable architecture for an algorithm or even for its part. Example of such use of the switch is the implementation of the Split and Merge algorithm, described in Chapter 8.

The availability of the DMA circuit in the Transputer makes it very suitable for use as a data routing module in a network. This suitability is further enhanced if Occam is used as a programming language. An important feature of Occam in this context is the ability to run a number of concurrent processes on a single Transputer. Thus each router procedure can be designed as a separate process; all the router processes can then run in parallel on the same Transputer. Furthermore, it is possible to design data router procedures specific to given architectures, independent of particular applications. The implementation of these concepts was described in detail in Chapter 5.

Experience gained during this project led to the conclusion that top-down modular approach is very important for the design of large-scale non-trivial parallel systems. Through this approach an insight can be gained as to the presence of parallel strains of different granularity in the system being implemented. This may then lead to better exploitation of parallelism and thus may result in more efficient parallel systems. This concept of modularity can be practically implemented through the use of several architectural 'blocks', where each block is a small network of processors. The arrangement of the blocks represents the design on the coarse level whereas the arrangement of the processors within the block represents the design on the fine level. This kind of arrangement have a number of advantages over the 'single architecture' approach. First of all it leads to a better design because the parallel implementation of a sub-task on a single block, and the arrangement of the blocks, may require very different parallel architectures. Secondly, it opens possibility for mixed-type architectures, where fine-level parallelism is implemented in hardware and coarse-level parallelism is implemented on the network level. In such a set-up a number of most versatile 'block' architectures could be identified (such as for example a 4x4 array or a pyramid of size 4) and each block could be fabricated as a single chip. This could improve the performance because the high speed of intercommunication inside the chip would decrease significantly communication overheads. This concept could also find a useful application in a multiuser supercomputer where a large number of such blocks would be available and each user would be able to configure the blocks to suit the application in hand.

Parallel processing plays on increasingly important role in the growth of the computing discipline as a whole. New parallel architectures and processing elements are continuously being developed and being brought into the reach of desk-top computing users. It is very important that a lot of research is directed towards methods for automatic extraction of parallel strains in existing programs (see, e.g. work by Kunii *et al* (1988)) so that the end-user can utilise the benefits of parallel processing without having to hand-craft each application.

References

Abdou, I. (1978), "Quantitative methods of edge detection", *USCIPI Report 830*, Image Processing Institute, University of Southern California.

Annaratone, M. *et al.* (1986), "WARP architecture and implementation", *the Proceedings of the 13th International Symposium on Computer Architecture.*

Arambepola, B. (1980), "Fast Computation of Multidimensional Discrete Fourier Transforms", *IEE Proceedings*, vol. 127, no. 1, pp. 49-52.

Ballard, D. H., and Brown, C. M. (1982), *Computer Vision*, Prentice-Hall, New Jersey.

Batcher, K. (1980), "Design of a massively parallel processor", *IEEE Trans. Computer*, vol. C-29, pp. 836-840.

Beetem, J., Denneau, M. and Weingarten, D. (1985), "The GF11 supercomputer", *the Proceedings of the 12th International Symposium on Computer Architecture*, pp. 108-118.

Binford, T. (1981), "Inferring surfaces from images", *Artificial Intelligence*, vol. 17, pp. 205-244.

Bouknight, W., Denenberh, S., Mclntyre, D., Randall, J., Sameh, A. and Slotnick, D. (1972), "The ILLIAC IV system", *Proceedings of IEEE*, vol. 60, no. 4, pp. 369-388.

Brice, C. and Fennema, C. (1970), "Scene analysis using regions", *Artificial Intelligence*, vol. 1, pp. 205-226.

Briggs, F. A., Hwang, K., and Fu, K. (1982), "PUMP: A shared-resource multiprocessor architecture for pattern analysis and image database management", in Preston, K. and Uhr, L. (eds), *Multicomputers and image processing-algorithms and programs*, Academic press, pp.319-330.

Brigham, E. (1974), *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs.

Brodatz, P. (1968), *Textures: A photographic album for artists and designers*, Reinhold, New York.

Brooks, R., Greiner, R. and Binford, T. (1978), "The ACRONYM model-based vision system", *Proceedings of the 4th IJCAI*, Kyoto, Japan, pp. 105-113.

Brown, C. (1989), "Reflexes in Computer Vision and their Control", *presented at BPRA Seminar*, Manchester University.

Burt, P., Anderson, C., Sinniger, O. and Van der Wal, G. (1986), "A pipeline pyramid machine", in Cantoni, V. & Levialdi, S. (eds), *Pyramidal Systems for Computer Vision*, Springer-Verlag, Berlin Heidelberg, pp. 133-152.

Canny, J. (1983), "Finding edges and lines in images", *MIT Artificial Intelligence Lab.*, Cambridge, MA, AI-TR-720.

Cantoni, V. and Levialdi, S (1983), "Matching the task to a computer architecture", *Computer Vision, Graphics and Image Processing*, vol. 22, pp. 301-339.

Cantoni, V. and Levialdi, S. (1987), "PAPIA: A case history", in Uhr, L., *Parallel Computer Vision*, New York, Academic Press.

Cantoni, V. and Levialdi, S. (1988), "Multiprocessor computing for images", *Proceedings of IEEE*, vol. 76, no. 8, pp. 959-969.

Cantoni, V., Ferretti, M., Levialdi, S. and Stefanelli, R. (1985), "PAPIA: Pyramidal architecture for parallel image analysis", *the Proceedings os the 7th IEEE Symposium on Computer Architecture*, (Urbana, II), pp. 237-242.

Chang, S. (1987), "Visual languages: A tutorial and survey", *IEEE Software*, vol.4, pp. 29-39.

Chen, P. and Pavlidis, T. (1979), "Segmentation by texture using a Co-occurrence matrix and a Split-and-Merge algorithm", *Computer vision, Graphics and Image Processing*, vol. 10, pp. 172-182.

Chien, Y. and Fu, K. S. (1974), "Recognition of X-ray picture patterns", *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-4, pp. 198-216.

Clark, M., Bovik, A. and Geisler, M. (1987), "Texture segmentation using Gabor modulation/demodulation", *Pattern Recognition Letters*, vol. 6, no. 4, pp. 261-267.

Conners, R. and Harlow, C. (1980), "A theoretical comparison of texture algorithms", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 3, pp. 204-222.

Conners, R., Trivedi, M. and Harlow, C. (1984), "Segmentation of a high-resolution urban scene using texture operators", *Computer vision, Graphics and Image Processing*, vol. 25, pp. 273-310.

Cooley, J. and Tukey, J. (1965), "An Algorithm for the Machine Calculation of Complex Fourier Series", *Mathematics of Computation*, vol. 19, no. 90, pp 297-301.

Daugman, J. (1985), "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters", *Journal of the Optical Society of America A*, vol. 2, pp. 1160-1169.

Davis, L., Rosefeld, A. and Weszka, J. (1975), "Region extraction by averaging and thresholding, ", *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-5, pp. 383-388.

Delp, E. and Mitchell, O. (1979), "Image compression using block truncation coding", *IEEE Trans. Communications*, vol. com-27, pp. 1335-1341.

Doherty, M., Bjorklund, C. and Noga, M. (1986), "Split-Merge-Merge: an enhanced segmentation capability", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami Beach, Florida, pp. 325-330.

Doherty, M., Noga, M. and Bjorklund, C. (1985), "Use of compound predicates in Split-and-Merge segmentation", *Proceedings of IEEE Computer Society Conference o Computer Vision and Pattern Recognition*, San Francisco, CA, pp. 659-662.

Douglass, R. J. (1982), "A pipeline architecture for image segmentation", *15th int. conference on system sciences*, pp. 360-367.

Duda, R. and Hart, P. (1973), *Pattern Classification and Scene Analysis*, Wiley-Interscience.

Dudgeon, D. and Mersereau, R. (1984), *Multidimensional Digital Signal Processing*, Prentice-Hall, New York.

Duff, M. j. B. (1978), "Review of the CLIP image processing system", *Proceedings of the AFIPS Conference*, vol. 47, NCC, pp. 1055-1060.

Eklundh, J. (1972), "A Fast Computer Method for Matrix Transposing," *IEEE Trans. Computers*, vol. C-21, no. 7, pp. 801-803.

Finkel, R (1987), "Large-Grain Parallelism -- Three Case Studies", in Jamieson, H., Gannon, D. and Douglass, R. (eds), The Characteristics of Parallel Algorithms, The MIT Press, pp. 21-63.

Flynn, M. (1966), "Very high speed computing", *Proceedings of the IEEE*, vol. 54, pp. 1901-1909.

Flynn, M. (1972), " Some computer organizations and their effectiveness", *IEEE Trans. Computer*, vol. C-21, pp. 948-960.

Fountain, T. (1986), "Array architectures for iconic and symbolic image processing", *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 24-33.

Fountain, T. (1987), *Processor Arrays: Architectures and Applications*, New York, Academic Press.

Fountain, T. J. (1981), "CLIP4: a progress report", in Duff, M. J. B. and Levialdi (eds), *Languages and architectures for image processing* , Acadmic press, London, pp. 283-291.

Fram, J. and Deutsch, E. (1975),"On the quantitative evaluation of edge detection schemes and their comparison with human performance", *IEEE Trans. Computers*, Vol. C-24, 6, pp. 616-628.

Fu, K. S. and Mui, J. (1981), "A survey on image segmentation", *Pattern Recognition*, vol. 12, pp. 395-403.

Gabor, D. (1946), "Theory of communication", *Journal of IEE*, vol. 93, pp. 429-457.

Galloway, M. (1975), “Texture analysis using gray level run lengths”, *CGIP*, vol. 4, pp. 172-179.

Gonzalez, R. and Wintz, P. (1988), *Digital Image Processing*, Addison Wesley, 2nd edition.

Granlund, G. (1978), "In search of a general picture processing operator", *Computer graphics and image processing*, vol. 8, pp. 155-173.

Granlund, G. (1980a), “Feedback processing of image information”, in Gelsema, E. & Kanal, L. (eds), *Pattern recognition in Practice*, North-Holland, pp. 45-59.

Granlund, G. (1980b), "GOP, a Fast and Flexible Processor for Image Analysis", *Proceedings of Fifth International Conference on Pattern Recognition*, pp. 489-492.

Haralick, R. (1979), "Statistical and structural approaches to texture", *Proceedings of the IEEE*, vol. 67, pp. 786-804.

Haralick, R. (1984), "Digital step edges from zero crossing of second directional derivatives", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 1, pp. 58-68.

Haralick, R. and Shapiro, L. (1985), "Image segmentation techniques", *Computer Vision Graphics*, vol. 29, pp. 100-132.

Haralick, R., Shanmugan, K. and Dinstein, I. (1973), “Textural features for image classification”, *IEEE Trans. System, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610-621.

Harp, J., Roberts, J. and Ward, J. (1985), "Signal Processing with Transputer arrays (TRAPS)", *Computer Physics Communications*, no. 37, pp. 77-86.

He, D., Wang, L. and Guibert, J. (1987), "Texture feature extraction", *Pattern Recognition Letters*, vol. 6, no. 4, pp. 269-273.

Hillis, W. (1985), *The Connection Machine*, Cambridge, MA: MIT Press.

Hockney, R. and Jesshope, C. (1988), *Parallel Computers 2*, Adam Hilger LTD, Bristol, England.

Horowitz, S. L., and Pavlidis, T. (1976), "Picture segmentation by a tree traversal algorithm", *JACM*, vol. 23, no.2, pp. 368-388.

Hough, P. (1962), "A method and means for recognizing complex patterns", *U. S. Patent 3*,069,654.

Huang, T. (1972), "Two-Dimensional Windows", *IEEE Trans. Audio and Electroacoustics*, vol. AU-20, no. 1, pp. 88-90.

Hubel, D. and Wiesel, T. (1974), "Sequence regularity and geometry of orientation columns in the monkey strait cortex", *J. Comp. Neural.*, vol. 158, pp. 267-293.

Hueckel, M. (1971), "An Operator which locates edges in digitised pictures", *JACM*, 18(1),pp. 113-125.

Hunt, D. J. (1981), "The ICL DAP and its application to image processing", in Duff, M. J. B. & Levialdi (eds), *Languages and architectures for image processing*, Academic press, London, PP. 275-282.

Hwang, K. and Briggs, F. A. (1985), *Computer architectures and parallel processing*, McGraw-hill.

Illingworth, J. and Kittler, J. (1988), "A survey of the Hough Transform", *Computer Vision Graphics, and Image Processing*, vol. 44, pp. 87-116.

INMOS Limited (1988a), *Occam 2 Reference Manual*, Prentice Hall.

INMOS Limited (1988b), *Transputer Development System*, Prentice Hall.

INMOS Limited (1989), *The Transputer Data Book*, INMOS document no. 72 TRN 203 01.

Kaiser, J. (1974), "Nonrecursive Digital Filter Design Using the I_0-sinh Window Function", *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 20-23.

Kaufmann, A (1967), *Graphs, dynamic programming, and finite games*, New York London: Academic Press.

Kittler, J and Duff, M. J. B. (eds) (1985), *Image processing system architectures*, Research studies press Ltd, England.

Knutsson, H., Post, B., and Granlund, G. (1980), "Optimization of arithmetic neighbourhood operations for image processing", *Proceedings of the 1st Scandinavian Conference on Image Analysis*, Sweden, pp. 87-92.

Kruse, B. (1973), "A parallel picture processing machine", *IEEE Trans. Computer*, vol. C-22, pp. 1075-1087.

Kruse, B., Gudmundsson, B. and Antonsson, D. (1980), "FIP: The PICAP II filter processor", *Proceedings of the 5th International Pattern Recognition Conference*, Miami Beach, Florida, pp. 484-488.

Kruse, B., Gudmundsson, B., and Antonsson, D. (1982), "PICAP and relational neighbourhood processing in FIP", in Preston, K. & Uhr, L. (eds), *Multicomputers and image processing-algorithms and programs* , Academic press, pp. 31-46.

Kuehn, J. and Siegel, H. (1986), "Simulation based performance measures for SIMD/MIMD processing", in Uhr, L., Levialdi, S., Preston, K., and Duff, M. (eds), *Evaluation of Multicomputers for Image Processing,* Academic press, New York, pp. 139-158.

Kuehn, J. and Siegel, H. , and Hallenbeck, P. (1982), "Design and simulation of an MC68000-based multimicroprocessor system", *International conference on parallel processing*, pp 353-362

Kunii, T., Nishimura, S., and Noma, T. (1988), "The Design of a Parallel System for Computer Graphics", *Presented in International Conference on Parallel Peocessing for Computer Vision and Display*, University of Leeds, UK.

Laprade, R. (1988), "Split-and-Merge segmentation of aerial photographs", *Computer vision, Graphics and Image Processing*, vol. 44, pp. 77-86.

Lee, C. H. (1986), "Recursive region splitting at hierarchical scope views", *Computer vision, Graphics and Image Processing*, vol. 33, pp. 237-258.

Levine, M. D. and Nazif, A. M. (1985a), "Rule-Based image segmentation: A dynamic control strategy approach", *Computer vision, Graphics and Image processing*, vol. 32, pp. 104-126.

Levine, M. D. and Nazif, A. M. (1985b), "Dynamic measurement of computer generated image segmentation", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-7, no. 2, pp. 155-164.

Levine, M. D. and Shaheen, S. I. (1981), "A modular computer vision system for picture segmentation and interpretation", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 5, pp. 540-556.

Li, H. and Maresca, M. (1987), "Polymorphic-torus network for computer vision", *IBM RC 12492*, Yorktown Heights.

Lodge, J. and Fahmy, M. (1980), "An Efficient lp Optimization Technique for the Design of Two-Dimensional Linear-Phase FIR Digital Filters", *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-28, no. 3, pp. 308-313.

Lougheed, D., McCubbrey, D. and Sternberg (1980), "Cytocomputers: Architectures for parallel image processing", *Proceedings of the Workshop Picture Data Description and Management*, Pacific Grove, CA, pp. 281-286.

Lui, W. (1983), "A New Pipelined/Parallel Architecture for Two Dimensional Fast Fourier Transform," *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pp. 214-219.

Machuca, R. and Gilbert, A. (1981), "Finding Edges in noisy scenes", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 1, pp. 103-112.

Maffei, L. and Fiorentini, A. (1973), "The visual cortex as a spatial frequency analyzer", *Vision Research*, vol. 13, pp. 1255-1267.

Manara, R. and Stringa, L. (1981), "The EMMA system: An industrial experience on a multiprocessor", in Duff, M. J. B. & Levialdi, S (eds), *languages and architectures for image processing* , Academic press, London, pp. 275-282.

Mansoor, W. M. and Claridge, E. (1989a), "2-Dimensional FFT Implementation on a Transputer Network", *Proceedings of the IASTED International Symposium on Applied Informatics*, Grindelwald, Switzerland, pp. 199-202.

Mansoor, W. M. and Claridge, E. (1989b), "2-Dimensional FFT Implementation on a Transputer Network", *Microcomputer Applications*, Vol. 8, No. 1, 1989.

Mansoor, W. M. and Claridge, E. (1989c), "Pyramid of Transputers for Texture Segmentation", *Proceedings of The Sixth Scandinavian Conference on Image Analysis*, Oulu, Finland, pp. 1200-1207.

Mansoor, W. M. and Sokolowska, E. (1988), "A Hierarchical Architecture for Image Segmentation", *Presented in The International Conference on Parallel Processing for Computer Vision and Display*, University of Leeds, UK.

Marcelja, S. (1980), "Mathematical description of the responses of simple cortical cells", *Journal of the Optical Society of America*,vol. 70, pp. 1297-1300.

Maresca, M., Lavin, M. and Li, H. (1988), "Parallel Architecture for Vision", *Proceedings of IEEE*, vol. 78, no. 8, pp. 970-981.

Marr, D. (1982), *Vision*, W. H. Freeman and Company, San Francisco.

Marr, D. and Hildreth, E. (1980), "Theory of Edge Detection", *Proc. Roy. Soc. London*, vol. B 207, pp. 187-217.

Marsan, M., Balbo, G., and Conte, G. (1986), *Performance Models of Multiprocessor System*, The MIT Press, Cambridge.

Martelli, A. (1976), "An application of Heuristic search methods to edge and contour detection", *Communications of the ACM*, vol. 19, no. 2, pp. 73-83.

May, D. (1983), "OCCAM", *SIGPLAN notices*, vol. 18, no. 4, pp. 69-79.

Mori, K., Kidode, M., Shinoddda, H. and Asada, H. (1978), "Design of local parallel pattern processor for image processing", *the proceedings of the AFIPS conference*, vol. 47, no. 17, NCC, pp. 1025-1031.

Muerle, J. and Allen, D. (1968), "Experimental evaluation of techniques for automatic segmentation of objects in a complex scene", in Cheng, G. et al. (eds), *Pictorial Pattern Recognition*, Thompson, Washington, D. C.

Nalwa, V., and Binford, T. (1986), "On detecting edges", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 699-714.

Nazif, A. M. and Levine, M. D. (1984), "Low level image segmentation: An expert system", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 5, pp. 555-577.

Nevatia, R. and Babu, R. (1980), "Linear feature extraction and description", *Computer Vision, Graphics, and Image Processing*, vol. 13, PP. 257-269.

Nilsson, N. (1971), *Problem-Solving Methods in Artificial Intelligence*, New York, McGraw-Hill.

Ohlander, R. (1975), *Analysis of natural scenes*, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh.

Ohlander, R., Price, K. and Reddy, D. (1978), "Picture segmentation using a recursive region splitting method", *Computer vision, Graphics and Image Processing*, vol. 8, pp. 313-333.

Petrou, M. and Kittler, J. (1988), "On the optimal edge detector", Proceedings of the fourth Alvey Vision Conference, University of Manchester, pp 191-196.

Pong, T., Shapiro, L. and Watson, L. (1984), "Experiments in segmentation using a facet model region grower", *Computer vision, Graphics and Image Processing*, vol. 25, pp. 1-23.

Powley, G., Kulick, J. and Gallerneault, C. (1986), "Automatic description of microstructures I. A study of segmentation techniques for the localization of microstructural regions", *J. Microscopy*, vol. 144, no. 2, pp. 127-136.

Pratt, K., Kane, J. and Andrews, H. (1969), "Hadamard Transform Image Coding", Proceedings of IEEE, vol. 57, no. 1, pp. 58-68.

Preston, K (1971), "Use of the Golay LOgic PRocessor in pattern recognition studies using hexagonal neighbourhood logic", in Fox, J. (ed), *Computers and automata*, New York, NY:Polytechnic Press, pp. 609-623.

Prewitt, J. (1970), "Object enhancement and extraction", in Lipkin, B. & Rosenfeld, A. (eds), *picture image processing and psychopictorics*, Academic press.

Rao, C. (1973), *Linear Statistical Inference and its Applications*, Wiley, New York.

Read, J. and Jayaramamurthy, S. (1972), "Automatic generation of texture feature detectors", *IEEE Trans. Computers*, vol. C-21, pp. 803-812.

Reddaway, S. (1978), "DAP-A flexible number cruncher", *Proceedings of the LASL workshop on Vector and Parallel Processors*, pp. 233-234.

Reddaway, S. (1980), "Revolutionary array processors", *the Proceedings of the 4th European Conference on Electrotechnics*, EUROCON80, Stutgart, pp. 730-734.

Reeves, A. (1984), "Survey: Parallel Computer Architectures for Image Processing", *Computer Vision, Graphics, and Image Processing*, vol. 25, pp. 68-88.

Reeves, A. (1985), "Multicluster: An MIMD system for computer vision", in Levialdi, S. (ed), *Integrated technology for parallel image processing*, Academic press, New York.

Reeves, A. (1986), "Pyramid algorithms on processor arrays", in Cantoni, V. & Levialdi, S. (eds), *Pyramidal Systems for Computer Vision*, Berlin, West Germany: Springer-Verlag, pp. 195-214.

Riseman, E., and Arbib, M. (1977), "Survey of the computational techniques of the visual segmentation of static scenes", *Computer Graphics and Image Processing*, vol.6, pp. 221-276.

Rivard, G. (1977), "Direct Fast Fourier Transform of Bivariate Functions," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-25, no. 3, pp. 250-252.

Roberts, L. (1965), "Machine reception of three-dimensional slides", In Tippet, J. *et al.* (eds), *optical and electronic-optical information processing* , Cambridge, MA, MIT press.

Rosenfeld, A. (1988), "Computer Vision: Basic Principles", *Proceedings of the IEEE*, vol. 76, no. 8, pp. 863-868.

Rosenfeld, A. and Kak, A. (1982), *Digital Picture Processing*, Academic press, vol. 1 & 2.

Rosenfeld, A. and Weszka, J. (1976), “Picture recognition and scene analysis”, *Computer*, vol. 9, pp. 28-38.

Samet, H. (1984), "Algorithm for the conversion of quadtree to rasters", *Computer Vision, Graphics, and Image Processing*, vol. 26, pp. 1-16.

Schaefer, D., Ho, P., Boyd, P. and Vallefos, C. (1987), "The GAM pyramid", in Uhr, L. (ed), *Parallel Computer Vision*, Orlando, FL: Academic Press, pp. 15-42.

Serra, J. (1982), *Image analysis and mathematical morphology*, Academic Press, London.

Shanmugan, K., Dickey, F., and Green, J. (1979), "An optimal frequency domain filter for edge detection in digital images", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 1, pp. 37-49.

Shaw, D. (1982), *The NON-VON supercomputer*, Computer Science Dept., Columbia University, New York.

Siegel, H., Siegel, L., Kemmerer, F., Mueller, J. and Smalley, J. (1981), "PASM: A Partitionable SIMD/MIMD system for image processing and pattern recognition", *IEEE Trans. Computers*, vol. C-30, pp. 934-937.

Siegel, L. (1981), "Image processing on a partitionable SIMD machine", in Duff, M. J. B. & Levialdi, S. (eds), *Languages and architectures for image processing*, Academic press, London, pp. 293-300.

Siegel, L., Siegel, H. and Swain, P. (1982), "Parallel algorithm performance measures", in Preston, K. and Uhr, L. (eds), *Multicomputers and Image Processing Algorithms and Programs*, Academic Press, pp. 241-252.

Sloan, K. R. (1977), *Word model driven recognition of natural scenes*, Ph. D. dissertation. The moor school of Elect. Eng. University of Pennsylvania, Philadelphia.

Spacek, L. (1984), *The computation of visual motion*, Ph. D. dissertation, university of Essex at Colchester.

Stansfield. S. (1986), "ANGY; a rule based expert system for automatic segmentation of coronary vessels from digital subtracted angiograms", *IEEE Trans. on pattern analysis and machine intelligence*, vol. PAMI-8, no. 2.

Sternberg, S. (1979), "Parallel Architecture for Image Processing", *Proceedings of the IEEE Computer Software and Applications Conference*, pp. 712-717.

Sternberg, S. (1983), "Biomedical Image Processing", *Computer*, vol. 16, no. 1, pp. 22-34.

Stout, Q. (1983), "Sorting, Merging, Selecting, and Filtering on Tree and Pyramid Machine", *Proceedings of the International Conference on Parallel Processing*, pp. 214-221.

Stout, Q. (1986), "Supporting Divide-and-Conquer Algorithms for Image Processing", *Journal of Parallel and Distributed Computing*, vol. 4, pp. 95-115.

Suciu, R. and Reeves, A. (1982), "A comparison of differential and moment based edge detectors", *IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pp. 97-102.

Synder, W. (1980), "Optimal computation of image gradients using eigenvector techniques and 3x3 neighborhood", *Proceedings of the 5th international conference on Pattern Recognition*, pp. 1314-1316.

Tanimoto, S. and Klinger, A. (eds) (1980), *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*, New York, Academic Press.

Tasto, M. and Wintz, P. (1971), "Image coding by adaptive block quantization", *IEEE Trans. Communications*, vol. COM-19, pp. 956-972.

Taylor, R. (1984), "Signal Processing with OCCAM and the Transputer," *IEE Proceedings*, vol. 131, no. 6, pp. 610-614.

Transtech Devices LTD (1988), *Transputer Motherboard and Module System Manual.*

Uhr, L. (1987), *Multicomputer Architectures for Artificial Intelligence: Towards Fast, Robust, Parallel Systems*, New York, Wiley.

Wermser, D. (1984), "Unsupervised segmentation by use of a texture gradient", *proceedings of 7th ICPR*, vol. 2, pp. 1114-1116.

Weszka, J., Dyer, C. and Rosenfeld, A. (1976), “Comparative study of texture measures for terrain classification”, *IEEE Trans. System, Man, and Cybernetics*, vol. SMC-6, pp. 269-285.

Yakimovsky, Y. (1976), "Boundary and Object Detection in Real World Images", *JACM*, vol. 23, no. 4, pp. 599-618.

Zucker, S. (1976), "Region growing: Childhood and adolescence", *Computer Graphics and Image Processing*, vol. 5, pp. 382-399.

Appendix A

This appendix describes theory of two-dimensional systems, methods of the filter design using the optimisation technique and the uncertainty theory.

A.1 Two Dimensional Discrete System

A 2-dimensional discrete system can be viewed as the process of applying an operator $\mathbf{L}$ to the input array $x(n, m)$ to get an output $y(n, m)$ where

$$y(m, n) = \mathbf{L}\,(x(m, n)). \tag{A.1}$$

This system is called linear when the following equation is valid for any constant c,

$$c\, y_1(m, n) + y_2(m, n) = \mathbf{L}(c\, x_1(m, n) + x_2(m, n)),$$

where $y_2(m, n) = \mathbf{L}(x_1(m, n))$, and $y_2(m, n) = \mathbf{L}(x_2(m, n))$. For the system to be shift-invariant, the following equality must be valid

$$y(m-m_0, n - n_0) = \mathbf{L}\,(x(m - m_0, n - n_0)),$$

for any values of m_0 and n_0.

It is very difficult to analyse and estimate the system performance by the response of equation (A.1). The best solution is to use a simple function like the impulse function as reference and study the response of the system for this function. This can be clarified by the following discussion.

The 2-dimensional impulse function $\delta(k, l)$ is defined as follows:

$$\delta(k, l) = \begin{cases} 1 & \text{for } k = 0 \;\&\; l = 0 \\ 0 & \text{otherwise} \end{cases}.$$

Then the 2-dimensional function x(m, n) can be evaluated by the following equation:

$$x(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x_{mn}\,\delta(m-k, n-l), \tag{A.2}$$

where x_{mn} is a constant and it is equal to x(m, n). Now by substituting equation (A.2) in equation (A.1),

$$y(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x_{mn}\,\mathbf{L}(\delta(m-k, n-l)). \tag{A.3}$$

Let h(k, n) be the impulse response of the system, then

$$h(m-k, n-l) = \mathbf{L}(\delta(m-k, n-l). \tag{A.4}$$

Now by substituting equation (A.4) in equation (A.3) the required result is achieved, which is

$$y(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x(m, n)\;h(m-k, n-l),$$

and this equation can be rewritten as follows:

$$y(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x(m-k, n-l)\;h(m, n). \tag{A.5}$$

This result means that the output of the system is the input array convolved with the impulse response of the system. Thus the problem of system analysis and design is to analyse and design the impulse response of the system. Similarly, filter design is the design of the impulse response of the filter.

In the image processing applications the input image is of a finite size, so the region of support G_{KL} of h(k,l) has to have finite limits, i.e., it is equal to zero outside that limits. This is quite useful from the stability point of view since the system is stable if the following inequality is valid

$$\sum_{k,l \in G} \sum |h(k, l)| < \infty .$$

This type of filter is called the finite impulse response (FIR) (nonrecursive) filter. It is clear now from the above discussion that the problem of design is to compute the impulse response of the system h(k, l).

A.2 Approximation Theory

The impulse response function h(n, m) can by described by a finite set of coefficients. The filter design problem is to calculate these coefficients.

Since the impulse response in the spatial domain is difficult to visualize, the best solution is to design its frequency response, because this is more understandable. Thus the problem of the filter design is now to determine the impulse response coefficients h(n, m) which produce a desired frequency response H(u, v). The relation between the impulse response and its frequency response is the Discrete Fourier Transform (DFT) relationship:

$$H(u, v) = \sum_{m=-N}^{N} \sum_{n=-N}^{N} h(m, n)\, W^{mu + nv}, \tag{A.6a}$$

$$h(m, n) = \sum_{u=-N}^{N} \sum_{v=-N}^{N} H(u, v)\, W^{-(mu + nv)}, \tag{A.6b}$$

where h(n, m) is zero outside a window of $(2N + 1)^2$ coefficients, and W is equal to $EXP(-j\,(\frac{2\pi}{2N + 1}))$, where j is $\sqrt{-1}$.

The above discussion has shown that it is necessary to select an ideal frequency response function and then get the impulse response by applying equation (A.6b). The result should be an impulse response with values equal to zero outside a certain region size. For all ideal filter functions this is impossible. The proof of this is out of the scope of this thesis but it can be found in other publications (e.g. Dudgeon & Mersereau, 1984). However, these filters can be realized by trying to design an impulse frequency response which approximates the desired frequency response. In other words the desired frequency response which cannot be realized should be approximated.

A.3 Design Methods

There are two main approaches for designing FIR filters; the non-optimal approaches and the optimal approaches (Dudgeon & Mersereau, 1984). The most common approach of the first type is the window method (Huang, 1972); for the second type the least square method (Lodge & Fahmy, 1980) is most commonly used.

A.3.1 The Window Method

This method based on the 1–dimensional (1D) window method. In this method an attempt is made to approximate the ideal impulse response rather than the ideal frequency response, i. e. it is a spatial domain method. To consider this in more detail, let $I(\omega_1, \omega_2)$ and $i(m, n)$ be the frequency response and the impulse response of the ideal filter respectively. Then the method is to find $h(m, n)$, which is the impulse response which approximates the ideal one $i(m, n)$, by applying the following equation:

$$h(m, n) = w(m, n)\, i(m, n),$$

where $w(m, n)$ is the window function of finite-extent region R,

$$R = \{(m, n): \ -M \leq m, n \leq M\},$$

so the approximated result $h(m, n)$ will be of the same finite-extent region R. The approximated frequency response $H(\omega_1, \omega_2)$ is related to the ideal frequency response $I(\omega_1, \omega_2)$ by the following equation:

$$H(\omega_1, \omega_2) = \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} I(\Omega_1, \Omega_2)\, W(\omega_1 - \Omega_1, \omega_2 - \Omega_2)\, d\Omega_1\, d\Omega_2 ,$$

where $W(\omega_1, \omega_2)$ is the frequency response of the window function, i. e., its Fourier transform. Since the desired filter behaviour is specified through the ideal frequency response rather than the impulse response, then the best way to implement this method is to chose the appropriate ideal frequency response and then take the inverse Fourier transform of it to get the ideal impulse response filter. Usually the region of support of i(m, n) is of infinite extent, so to minimize the aliasing error the extent of the region of support of the inverse Fourier transform should be several times larger than the extent of the R, and may be called G.

There are three requirements for the choice of the window function (Huang, 1972). Firstly, it must have the region of support R. Secondly, $W(\omega_1, \omega_2)$ should approximate a 2–dimensional (2D) impulse function. Thirdly, if h(m, n) is to be zero-phase, the window function should be zero-phase as well. These requirements are the same as for the 1D case, so a good window function for the 1D case should suggest a good one for the 2D case. The relationship between the two could be expressed by either of the following equations:

$$w_r(m, n) = w_1(m)\, w_2(n),$$

or

$$w_c(m, n) = w(\sqrt{m^2 + n^2}),$$

where w_r is a 2D window with a square or rectangular region of support, w_c is a 2D window which has nearly circular region of support, and w, w_1, and w_2 are 1D windows. If the 1D window is good for the 1D filter design the 2D windows will be good as well. Example of such windows are the rectangular window:

$$w_r(m, n) = \begin{cases} 1, & (m, n) \in R \\ 0, & \text{otherwise} \end{cases}$$

and the Kaiser window (Kaiser, 1974)

$$wr(m, n) = \begin{cases} \dfrac{I_o(\alpha\sqrt{1 - \left(\frac{m}{M}\right)^2})\, I_o(\alpha\sqrt{1 - \left(\frac{n}{M}\right)^2})}{I_o^2(\alpha)}, & (m, n) \in R \\ 0, & \text{otherwise} \end{cases}$$

where $I_0(x)$ is the modified Bessel function of the first kind of order zero.

A.3.2 Least-square Design Method

The filter design problem is to calculate filter coefficients of the desired frequency response $H(\omega_1, \omega_2)$ which approximate the ideal frequency response $I(\omega_1, \omega_2)$. Let E be the error between these two functions. It can be evaluated by the following equation:

$$E = H(\omega_1, \omega_2) - I(\omega_1, \omega_2).$$

So the problem now is to calculate the coefficients of the filter to minimize the L2–norm (mean-square value) of E,

$$E_2 = \frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} \left|E(\omega_1, \omega_2)\right|^2 d\omega_1\, d\omega_2 . \tag{A.7}$$

There are several ways to solve the problem; three of them will now be considered. The function E_2 can be evaluated in the spatial domain by using Parseval's theorem,

$$E_2 = \sum_{m,n \in G}\sum [h(m, n) - i(m, n)]^2,$$

since h(m, n) is equal to zero outside region R, then

$$E_2 = \sum_{m,n \in R}\sum [h(m,n) - i(m,n)]^2 - \sum_{m,n \notin R}\sum i^2(m,n). \qquad (A.8)$$

Since both summations of equation A.8 are positive and the coefficient values affect the first one then the optimal solution which is the minimum value of E2 is found when

$$h(m,n) = \begin{cases} i(m,n), & (m,n) \in R \\ 0, & (m,n) \notin R \end{cases}.$$

This solution is the same as that of the window method with a constant window over R.

In the slightly more general case where linear constraints may be present, the following method may follow. The frequency response is given by

$$H(\omega_1, \omega_2) = \sum_{m,n \in R}\sum h(m,n)\, \mathrm{EXP}[-j\omega_1 m - j\omega_2 n]$$

For real filter function (zero phase),

$h(m, n) = h(-m, -n)$,

$H(\omega_1, \omega_2)$ can be evaluated by the following equation:

$$H(\omega_1, \omega_2) = h(0,0) + \sum_{m,n \in R'}\sum 2h(m,n)\cos(\omega_1 m + \omega_2 n), \qquad (A.9)$$

the limits of summation are in the region R' which is half of the region R. It is therefore convenient to put equation A.9 in the following form:

$$H(\omega_1, \omega_2) = \sum_{i=1}^{F} a(i)\, \phi_i(\omega_1, \omega_2), \qquad (A.10)$$

where i is an index that denotes some ordering of the samples (m, n) in region R',

$a(i) = h(m, n)$,

F is number of coefficients of h, and

$$\phi_i(\omega_1, \omega_2) = \begin{cases} 2\cos(\omega_1 m + \omega_2 n) & (m, n) \neq (0, 0) \\ 1, & (m, n) = (0, 0) \end{cases}$$

E2 can be evaluated by the substitution of equation A.10 in equation A.7,

$$E_2 = \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left[\sum_{i=1}^{F} a(i)\, \phi_i(\omega_1, \omega_2) - I(\omega_1, \omega_2) \right]^2 d\omega_1\, d\omega_2 . \qquad \text{(A.11)}$$

Then E2 can be minimized by taking its derivative with respect to each coefficient of a(k), setting these derivatives to zero and solve the resulting equations. Since the partial derivatives [∂E2/∂a(k)] are all linear functions of the unknown coefficients, this requires at most the solution of F linear equations. These equations can be written as

$$\sum_{i=1}^{F} a(i)\, \phi_{ik} = I_k , \qquad \text{(A.12)}$$

where

$$a(i) = h(m, n),$$

$$\phi_{ik} = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \phi_i(\omega_1, \omega_2)\, \phi_k(\omega_1, \omega_2)\, d\omega_1\, d\omega_2 ,$$

$$I_k = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} I(\omega_1, \omega_2)\, \phi_k(\omega_1, \omega_2)\, d\omega_1\, d\omega_2 . \qquad \text{(A.13)}$$

In the frequently occurring special case that $\phi_i(\omega_1, \omega_2)$ are orthogonal, $\phi_{ik} = 0$ for $i \neq k$, the solution of equation A.12 is simply

$$a(i)\, \phi_{ii} = I_i .$$

The number of degrees of simultaneous linear equations to be solved places an effective upper limit on the number of degrees of freedom that can be accommodated.

The third method of coefficient evaluation uses a modified error measure. The error measure E_2 weights errors equally at all frequencies. The filters which are designed to minimize E_2 may not always be satisfactory; they suffer from large passband and stopband ripple. The integral of equation A.12 may also be difficult to evaluate if $I(\omega_1, \omega_2)$ is not simple. These problems can be partially alleviated by replacing the error E2 by E'_2 where

$$E'_2 = \sum_{c=1}^{C} W_c \left[H(\omega_{1c}, \omega_{2c}) - I(\omega_{1c}, \omega_{2c}) \right]^2, \qquad (A.14)$$

where $(\omega_{1c}, \omega_{2c})$ are the constraint frequencies, corresponding to a finite number of discrete locations C in the 2D frequency plane, and the (positive) numbers W_c denote weighting values.

Finding the coefficients a(k) that minimize E2' involves the solution of F linear equation in F unknowns given by

$$\sum_{i=1}^{F} a(i)\, \phi_{ik} = I_k, \quad k = 1,\ldots,F \qquad (A.15)$$

where

$$\phi_{ik} = \sum_{c=1}^{C} W_c\, \phi_i(\omega_{1c}, \omega_{2c})\, \phi_k(\omega_{1c}, \omega_{2c}),$$

$$I_k = \sum_{c=1}^{C} W_c\, I(\omega_{1c}, \omega_{2c})\, \phi_k(\omega_{1c}, \omega_{2c}).$$

A.4 Uncertainty Principle

There are two schools of opinions (Daugman, 1985) in describing the fundamental character of early visual representation, the first say that it involves the space-domain local feature detection (Hubel & Wiesel, 1974), the second (Maffei & Fiorentini, 1973) say that it more closely resembles a Fourier-like decomposition into spatial-frequency components. The necessity of this finding is to design a vision system which can process efficiently a vast amount of image information and this can be achieved by extracting and representing the image with optimal economy. The work by Marcelja (1980) affirmed that the visual mechanism is deemed linear, and then its selectivities in either domain imply complementary limits in the other. This problem is called the uncertainty problem and it is well defined by Gabor (1946) who pointed it out with respect to the 1-dimensional (1D) signal. Gabor said that a signals' specificity simultaneously in time and frequency is fundamentally limited by a lower bound of the product of its frequency band width and time duration. If the filter weighting function is denoted f(t), a standard measure of effective width (Δt) is given by the second moment of its energy distribution:

$$(\Delta t)^2 = \frac{\int_{-\infty}^{\infty} t^2 f(t)\, f^*(t)\, dt}{\int_{-\infty}^{\infty} f(t)\, f^*(t)\, dt},$$

where f*(t) is the complex conjugate of f(t).

Similarly in the frequency domain, F(w) is Fourier transform of f(t) and Δw is the effective bandwidth,

$$(\Delta w)^2 = \frac{\int_{-\infty}^{\infty} w^2 F(w)\, F^*(w)\, dw}{\int_{-\infty}^{\infty} F(w)\, F^*(w)\, dw}.$$

Then the 1D uncertainty principle specifies a fundamental lower bound on the possible values of their product:

$$(\Delta t)\,(\Delta w) \geq \frac{1}{4\pi}. \qquad (A.16)$$

Gabor found the general family of functions that achieve the theoretical lower limit of the uncertainty principle. These are of the following form:

$$f(t) = \mathrm{EXP}\left[\frac{-(t - t_o)^2}{\alpha^2} + jwt\right] , \qquad (A.17)$$

which in complex notation describes the modulation product of a sine wave with arbitrary frequency (w) and a Gaussian envelope of arbitrary duration occurring at interval t_o.

Later Daugman (1985) suggested a 2D model for the uncertainty principle. Let f(x, y) be the filter function and F(u, v) the Fourier transform of it. Then there are several possible generalizations of the concept of effective width, due to the fact that there are three Cartesian second moments for the energy distribution f(x, y) f*(x, y):

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^2 f(x, y)\, f^*(x,y)\, dx\, dy \ ,$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y^2 f(x, y)\, f^*(x,y)\, dx\, dy \ ,$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x\, y\, f(x, y)\, f^*(x,y)\, dx\, dy \ .$$

The first two moments compute the variance around the y and x axes respectively, and these are equal to the square of the distributions of effective width Δx and effective length Δy respectively; The third one relates to the skew moments and computes asymmetry of the diagonals. The 2D distribution can be rotated so that its skew moment is zero and the angle of this rotation determines the distribution's principle axes.

There are two uncertainty principles constraining the effective width Δx and the effective length Δy of a 2D filter f(x, y), and the effective width Δu and the effective length Δv for its Fourier transform F(u, v). Generally for any arbitrary f(x,y) centred at (x_o, y_o), whose 2D Fourier transform F(u, v) is centred at (u_o, v_o), the following two uncertainty principles apply:

$$(\Delta x)\,(\Delta u) \geq \frac{1}{4\pi}\,, \tag{A.18a}$$

$$(\Delta y)\,(\Delta v) \geq \frac{1}{4\pi}\,, \tag{A.18b}$$

where

$$(\Delta x) = \left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x - x_o)^2\, f(x, y)\, f^*(x,y)\, dx\, dy \Big/ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\, f^*(x,y)\, dx\, dy\right]^{\frac{1}{2}},$$

$$(\Delta u) = \left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (u - u_o)^2\, F(u, v)\, F^*(u,v)\, du\, dv \Big/ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u, v)\, F^*(u,v)\, du\, dv\right]^{\frac{1}{2}},$$

$$(\Delta y) = \left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (y - y_o)^2\, f(x, y)\, f^*(x,y)\, dx\, dy \Big/ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\, f^*(x,y)\, dx\, dy\right]^{\frac{1}{2}},$$

$$(\Delta v) = \left[\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (v - v_0)^2 F(u, v) F^*(u,v)\, du\, dv \Big/ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) F^*(u,v)\, du\, dv \right]^{\frac{1}{2}} .$$

If these functions and transforms are rotated out of such axes, the products of effective widths will in general increase. If the principal axes do correspond, then each of the products (Δx) (Δu) and (Δy) (Δv) represents the effective area occupied by the filter in the corresponding 2D domain. The joint resolution that can be achieved by any 2D filter in the two 2D domains is constrained by a fundamental lower limit:

$$(\Delta x)\, (\Delta y)\, (\Delta u)\, (\Delta v) \geq \frac{1}{16\pi^2} . \qquad \text{(A.19)}$$

This represent the new uncertainty principle which expresses the theoretical limit of joint 2D resolution in the two domains.

Daugman (1985) has presented a family of functions f(x, y) and their 2D Fourier transforms F(u, v) which achieve the lower bound in equation (A.19);

$$f(x, y) = g(x, y)\, \mathrm{EXP}\{- 2\pi j[u_o\, x' + v_o\, y']\} , \qquad \text{(A.20)}$$

$$F(u, v) = G(u, v)\, \mathrm{EXP}\{- 2\pi j[x_o\, u' + y_o\, v']\} , \qquad \text{(A.21)}$$

where

$$g(x, y) = \mathrm{EXP}\{- \pi[x'^2 a^2 + y'^2 b^2]\},$$

$$G(u, v) = \mathrm{EXP}\{- \pi[u'^2/a^2 + v'^2/b^2]\},$$

which are Gaussian functions, $x' = x - x_o$, $y' = y - y_o$, $u' = u - u_o$, $v' = v - v_o$, and a and b are the space parameters defining the scale of the Gaussian function along both axes.

The filter function proposed by Granlund (1978b) in equation (7.1) belongs to the above family, but he used different derivation to satisfy the lower bound of the uncertainty principle. However, Daugman derivation is more precise and constructive because his results are more general and he put it in straight language that the family of this type of function is the optimal solution for uncertainty principle. Granlund did not mention Gabor's (1946) work at all, and it is known that Gabor was the first person who tackled the uncertainty problem and solved it for the 1D case.

Appendix B

This appendix shows the code for the main procedure in the host Transputer which its Occam model was shown in figure 5.2 page 82. The code printed here is for the procedure to communicate with the network for implementing the FFT RC algorithm. The code for the procedures used to communicate with networks for implementing Split & Merge and FFT VR algorithms are similar with very small changes.

```
PROC host   (CHAN OF INT keyboard,CHAN OF ANY screen,
            from.filer,to.filer,
            CHAN OF grid app.in,app.out,
            VAL BOOL using.subsystem)
  CHAN OF inter cont.in.filing,cont.in.sc,cont.out.sc,filing.sc:
  CHAN OF grid filing.cont.in,filing.cont.out,cont.in.out,
        hang.free.filing,filing.hang.free:
  CHAN OF ANY echo:
  #USE tfftlib -- It is the library which contains procedures for implementing FFT
RC
  #USE userio
  PROC screen.hnd(CHAN OF ANY screen,echo,
              CHAN OF inter cont.in.sc,cont.out.sc,
              filing.sc)
    BYTE char,length:
    [100]BYTE string:
    INT number:
    BOOL continue:
    SEQ
      continue:=TRUE
      WHILE continue
```

```
ALT
  cont.in.sc ? CASE
    sc.int;number
      SEQ
        write.int(screen,number,0)
        newline(screen)
    sc.string;length::string
      SEQ
        number:=INT length
        write.len.string(screen,number,string)
    f.terminate
      continue:=FALSE
  filing.sc ? CASE
    sc.int;number
      SEQ
        write.int(screen,number,0)
        newline(screen)
    sc.string;length::string
      SEQ
        number:=INT length
        write.len.string(screen,number,string)
    f.terminate
      continue:=FALSE
  cont.out.sc ? CASE
    sc.int;number
      SEQ
        write.int(screen,number,0)
        newline(screen)
    sc.string;length::string
      SEQ
        number:=INT length
        write.len.string(screen,number,string)
    f.terminate
     INT any:
     SEQ
      continue:=FALSE
   filing.sc ? CASE
    sc.int;number
     SEQ
      write.int(screen,number,0)
      newline(screen)
    sc.string;length::string
     SEQ
      number:=INT length
      write.len.string(screen,number,string)
    f.terminate
     continue:=FALSE
   echo ? char
    screen ! char
:
#USE tfftlib
#USE filerhdr
#USE krnlhdr
#USE interf
#USE userio
#USE extrio
#USE t4math
PROC filing (CHAN OF INT keyboard,CHAN OF ANY echo,
      to.filer,from.filer,
   CHAN OF grid  filing.cont.out,filing.cont.in,
       hang.free.filing,filing.hang.free,
   CHAN OF inter cont.in.filing,filing.sc)
  VAL kernel.size IS 3:
  VAL block.size IS 512:
```

```
[block.size*257]BYTE image.data:
[20]INT t:
INT echo1,n,lo,e,N ,NN,k,letter,so,P,Tp1,Tc,Tp2,display:
INT any, result,record.length,len, block.no,convolution:
INT time,start.time,finish.time,time.of.processing:
VAL max IS 70000:
VAL Pn1 IS [0]: --for single transputer
VAL Pn4 IS [0,4,8,12]:--for array of size four
  VAL Pn16 IS [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]:--for array of size
sixteen
[max]BYTE x:
[40]BYTE string:
BYTE length:
INT image.size,sub.image.size,k1,k2,start,end,size:
REAL32 real,outreal:
[23]BYTE dos.name:
BOOL  send.data,continue,file.dosenot.exist,un.accepted.value,
    data.recieved,another,process.finished:
TIMER clock:
PROC send.to.screen(VAL []BYTE strings)
 --VAL strings IS "*c*ntype kernel.size*c*n":
 BYTE length:
 SEQ
  length:=BYTE (SIZE strings)
  filing.sc ! sc.string;length::strings
:
PROC open.read (VAL []BYTE filename, INT len,result)
 --This procedure to open dos file for read
 INT command :
 SEQ
  to.filer ! tkf.open.read
  to.filer ! len; filename
```

```
    from.filer ? command; result
  :
  PROC open.write (VAL []BYTE filename, INT len,result)
  --This procedure to open dos file for write
  INT command :
    SEQ
      to.filer ! tkf.open.write
      to.filer ! len; filename
      from.filer ? command; result
  :
  PROC read (INT result,INT record.length,[]BYTE record)
    --This procedure to read a record of record.length BYTES long from opened
dos file
    INT command:
    SEQ
      to.filer ! tkf.read
      from.filer ? command; result
      from.filer ? record.length::record
  :
  PROC write (VAL []BYTE record,INT result)
    --This procedure to write a record of BYTES to the opened dos file
    INT command:
    SEQ
      to.filer ! tkf.write
      to.filer ! SIZE record;record
      from.filer ? command; result
  :
  PROC close (INT result, INT len, []BYTE filename)
    --This procedure to close the opened dos file
    INT command:
    SEQ
      to.filer ! tkf.close
```

```
 from.filer ? command; result
 from.filer ? len::filename


:

PROC input.data(CHAN OF ANY echo,[]BYTE image.data,INT image.size)
 --This procedure to input  data of type BYTES from a dos file which its name to
 -- be typed in by the user
 VAL block.size IS 512:
 INT time, result,record.length,len, block.no:
 INT any:
 [abs.id.size]BYTE dos.name:
 [30]BYTE inputfile.name:
 BOOL file.doesnot.exist:
 TIMER clock:
 SEQ
   file.doesnot.exist:=TRUE,
   WHILE file.doesnot.exist
     SEQ
       block.no := 0
       send.to.screen ("*c*ntype file name*c*n")
        any:=INT ' '
       read.echo.text.line(keyboard,echo,len,inputfile.name,any)
       len:=len-1
       open.read (inputfile.name,len, result)
       send.to.screen("*c*nresult from open*c*n")
       filing.sc ! sc.int;result
       IF
         result = 0
           SEQ
             record.length := block.size
             WHILE record.length = block.size
               INT start:
```

```
      SEQ
        start:=  block.no * block.size
        record IS [image.data FROM start FOR block.size]:
        SEQ
          read (result, record.length, record)
          block.no := block.no +1
    image.size:= (block.size*(block.no -1))+record.length
    send.to.screen("*c*nimage.size=")
    filing.sc ! sc.int;image.size
    send.to.screen(" bytes read*c*n")
    close (result, len, inputfile.name)
    send.to.screen("*c*nresult from close*c*n")
    filing.sc ! sc.int;result
    file.doesnot.exist:=FALSE
  TRUE
    send.to.screen("*c*nfile does not exist try again*c*n")

:

PROC output.data(CHAN OF ANY echo,[]BYTE image.output,INT image.size)
  --This procedure to write  data of type BYTES onto a dos file which its name to
  -- be typed in by the user
  VAL block.size IS 512:
  INT time, result,record.length,len, block.no:
  INT any:
  [abs.id.size]BYTE dos.name:
  [30]BYTE outputfile.name:
  TIMER clock:
  SEQ
    send.to.screen("type outputfile name")
    any:=INT ' '
    read.echo.text.line(keyboard,echo,len,outputfile.name,any)
    send.to.screen("Trying to open file*c*n")
```

```
len := len-1
open.write (outputfile.name,len, result)
send.to.screen("Result from open file command : ")
filing.sc ! sc.int;result
newline (echo)
block.no := 0
IF
  image.size <= block.size
    SEQ
      record IS [image.output FROM 0 FOR image.size]:
      write (record,result)
      send.to.screen("Buffer read from file - result ")
      filing.sc ! sc.int;result
      newline (echo)
      block.no := block.no +1
  TRUE
    SEQ
      WHILE (image.size-(block.no * block.size)) > block.size
        VAL start IS block.no * block.size:
        record IS [image.output FROM start FOR block.size]:
        SEQ
          write (record,result)
          block.no := block.no +1
      VAL start IS block.no * block.size:
      record IS [image.output FROM start FOR (image.size-(block.no *
       block.size))]:
      SEQ
        write (record,result)
        block.no := block.no +1
  filing.sc ! sc.int;block.no
  send.to.screen("  Blocks where write *c*n")
  result:=((block.size*(block.no -1))+record.length)
```

```
 filing.sc ! sc.int;result
 send.to.screen(" Bytes where read*c*n")
 close (result, len, dos.name)
 send.to.screen("result from close ")
 filing.sc ! sc.int;result
 newline(echo)
 write.len.string (echo,len,dos.name)
:
 PROC input.real.data(CHAN OF ANY echo, []REAL32 real.data, INT
data.size)
 --This procedure to input  data of type REAL from a dos file which its name to
 -- be typed in by the user
 INT i,j,ik:
 BOOL error,run:
 [100000]BYTE data:
 [30]BYTE block:
 SEQ
  send.to.screen("The following type in the line kernels file*c*n")
  input.data(echo,data,data.size)
  SEQ
   i:=0
   j:=0
   run:=TRUE
   WHILE i< data.size
    SEQ
     ik:=0
     WHILE ((data[i]>='0')AND(data[i]<='9'))OR
         (data[i]='E')OR
         (data[i]='+')OR
         (data[i]='-')OR
         (data[i]='.')
      SEQ
```

```
            block[ik]:=data[i]
            i:=i+1
            ik:=ik+1
        IF
          ik=0
            i:=i+1
          TRUE
            VAL blocke IS [block FROM 0 FOR ik]:
            SEQ
              STRINGTOREAL32(error,real.data[j],blocke)
              j:=j+1
      data.size:=j
:
PROC distribute.data([]BYTE x,image.data,
                     INT N,P,k1)
  --this procedure distribute image to P transputers
  -- the first N/P rows to the 1st one an the second
  --N/P rows to the second transputer and so on
  INT F,k,sub.image.size:
  SEQ
    F:=N/P
    SEQ i1=0 FOR P
      SEQ
        k:=0
        SEQ i=i1*F FOR F
          SEQ j=0 FOR N
            SEQ
              x[k]:=image.data[j+(i*N)]
              k:=k+1
        sub.image.size:=k
        IF
          k1=0
      filing.cont.out ! w.data;Pn1[i1];sub.image.size::x;F;N
     k1=1
      filing.cont.out ! w.data;Pn4[i1];sub.image.size::x;F;N
     TRUE
      filing.cont.out ! w.data;Pn16[i1];sub.image.size::x;F;N
 :
PROC distribute.real.data([]REAL32 image.data,INT N,P,k1)
 --this procedure distribute image of REAL32 data type to P transputers for FFT
RC
 -- the first N/P rows to the 1st one an the second
 --N/P rows to the second transputer and so on
 INT F,k,sub.image.size:
 [4200]REAL32 x:
 SEQ
  F:=N/P
  SEQ i1=0 FOR P
   SEQ
    k:=0
    SEQ i=i1*F FOR F
     SEQ j=0 FOR N
      SEQ
       x[k]:=image.data[j+(i*N)]
       k:=k+1
    sub.image.size:=k
    IF
     k1=0
      filing.cont.out ! real.data;Pn1[i1];sub.image.size::x;F;N
     k1=1
      filing.cont.out ! real.data;Pn4[i1];sub.image.size::x;F;N
     TRUE
      filing.cont.out ! real.data;Pn16[i1];sub.image.size::x;F;N
 :
```

```
[12]BYTE inputfile.name :
[12]BYTE outputfile.name :
SEQ
 continue :=TRUE
 WHILE continue
  SEQ
   send.to.screen("*c*ntype 0 for display else for not to display*c*n")
    echo1:=INT ' '
   read.echo.int(keyboard,echo,display,echo1)
   input.data(echo,image.data,image.size)
   filing.cont.out ! display.data;image.size::image.data
   NN:=image.size
   real:=REAL32 TRUNC NN
   outreal:=SQRT(real)
   N:=INT TRUNC outreal
   un.accepted.value:=TRUE
   WHILE un.accepted.value
     SEQ
      VAL string IS "*c*ntype 2 for convolution or 1 for fft*c*n":
      SEQ
       length:=BYTE (SIZE string)
        filing.sc ! sc.string;length::string
       echo1:=INT ' '
      read.echo.int(keyboard,echo,convolution,echo1)
      VAL string IS "*c*n":
      SEQ
       length:=BYTE (SIZE string)
        filing.sc ! sc.string;length::string
      un.accepted.value:=FALSE
      IF
       (convolution = 1) OR (convolution=2)
         un.accepted.value:=FALSE
   TRUE
     send.to.screen("*c*ntype in 1 or 2 only, try again*c*n")

send.to.screen( "*c*ntype k1 where P=4 to power k1*c*n")
echo1:=INT ' '
read.echo.int(keyboard,echo,k1,echo1)
clock ? time
clock ? AFTER time PLUS 50000
filing.cont.out ! conv;convolution
k2:=INT TRUNC POWER(REAL32 TRUNC 2,REAL32 TRUNC k1)
P:=k2*k2
filing.cont.in ! net.s;P;display
sub.image.size:=N/P
send.to.screen("*c*n sub.image.size=*c*n")
filing.sc ! sc.int;sub.image.size
IF
  convolution=2
    [4200]REAL32 real.data:
    INT data.size:
    SEQ
      input.real.data(echo, real.data, data.size)
      distribute.real.data( real.data, N,P,k1)
  TRUE
    SKIP
any:=0
send.to.screen("*c*n press any to send value n to controller*c*n")
keyboard ? any
send.to.screen("Processing is going on")
so:=N/k2    --this is because P=k2*k2
clock ? start.time
--distribute.data.VR( x,image.data, N,so,k1)
distribute.data( x,image.data,N,P,k1)
```

```
process.finished:=FALSE
WHILE NOT process.finished
 ALT
  cont.in.filing ? CASE
   oh;n::t
    SKIP
   B;process.finished
    SEQ
     clock ? finish.time
     time.of.processing := ((finish.time-start.time)*64)/1000
     send.to.screen("*c*n  total time.of.processing ( msec)=")
     filing.sc ! sc.int;time.of.processing
     SEQ i=0 FOR 3
      SEQ
       time.of.processing := ((t[i])*64)/1000
       send.to.screen("*c*n  time.of.processing ( msec)=")
       filing.sc ! sc.int;time.of.processing
     send.to.screen("*c*nprocessing complete")
     send.to.screen("type a for anotheror f to ex*c*n")
     another:=TRUE
     WHILE another
      SEQ
       keyboard ? letter
       IF
        letter=(INT 'a')
         SEQ
          another:=FALSE
        letter=(INT 'f')
         SEQ
          continue:=FALSE
          another:=FALSE
          filing.cont.out ! w.terminate
```

```
              filing.cont.in ! w.terminate
              filing.hang.free ! w.terminate
            TRUE
             SEQ
              send.to.screen("*c*ntry again")
      hang.free.filing ? CASE
       exe.terminate
        SEQ
         process.finished:=TRUE
         continue:=FALSE
         filing.cont.out ! exe.terminate
         filing.cont.in ! exe.terminate
:
#USE tfftlib
#USE t4math
#USE filerhdr
#USE interf
#USE userio
PROC input.controller (CHAN OF grid app.out,
                  filing.cont.in,cont.in.out,
                  CHAN OF inter cont.in.filing,cont.in.sc)
 PROC send.to.screen(VAL []BYTE strings)
  --VAL strings IS "*c*ntype kernel.size*c*n":
  BYTE length:
  SEQ
   length:=BYTE (SIZE strings)
   cont.in.sc ! sc.string;length::strings
 :
 VAL Pn1 IS [0]:
 VAL Pn4 IS [0,1,2,3]:
  VAL Pn16 IS [0,2,8,10,1,3,9,11,5,7,13,15,4,6,12,14]:
  INT P,lo,e,N ,NN,Pn,k,no.of.results,Tp1,Tc,Tp2:
```

```
[20]INT t:
INT n,display:
VAL max IS 65600:
VAL fone IS 0.1(REAL32):
REAL32 temp:
[max]INT result.out:
[max]BYTE data:
[130][130]REAL32 x,xi:
INT any,time, result,len:
BYTE letter,length:
INT start,end,size,time.of.processing:
TIMER clock:
BOOL  continue,data.recieved:
BOOL continue,not.halt,processing,another,process.finished:
SEQ
  processing:=TRUE
  no.of.results:=0
  clock ? start
  WHILE  processing
    ALT
      app.out ? CASE
        c.akn;Pn;data.recieved
          SEQ
            send.to.screen("*c*ndata recieved for Pn=")
            cont.in.sc ! sc.int;Pn
        c.data;Pn;size::result.out
          SEQ
            IF
              P=4
                INT map:
                SEQ i=0 FOR size
                  SEQ
          map:=Pn4[(Pn/4)]
          map:=(map*size)+i
          IF
            ABS(REAL32 TRUNC result.out[i]) > (REAL32 TRUNC 255)
              data[map]:= BYTE 255
            TRUE
              data[map]:=
                   BYTE(INT TRUNC (ABS(REAL32 TRUNC result.out[i])))
      TRUE
        SEQ i=0 FOR size
          IF
            ABS(REAL32 TRUNC result.out[i]) > (REAL32 TRUNC 255)
              data[((Pn*size)+i)]:= BYTE 255
            TRUE
              data[((Pn*size)+i)]:=
                   BYTE (INT TRUNC (ABS(REAL32 TRUNC result.out[i])))
      no.of.results:=no.of.results+1
      IF
        no.of.results=P
          SEQ
            IF
              display = 0
                cont.in.out ! display.data;(P*size)::data
              TRUE
                SKIP
            no.of.results:=0
            IF
              P=4
                SEQ
                  SEQ i=0 FOR N
                    SEQ j=0 FOR N
                      SEQ
                xi[i][j] :=fone
                x[i][j] :=fone
              vr2dfftl(x,xi,N)
            TRUE
              SKIP
            process.finished:=TRUE
            cont.in.filing ! B;process.finished
          TRUE
            SKIP
      c.data.oh;Pn;size::result.out;n::t
        SEQ
          cont.in.filing ! oh;n::t
          SEQ i=0 FOR size
            IF
              ABS(REAL32 TRUNC result.out[i]) > (REAL32 TRUNC 255)
                data[i]:= BYTE 255
              TRUE
                data[i]:=BYTE (INT TRUNC (ABS(REAL32 TRUNC
result.out[i])))
          no.of.results:=no.of.results+1
          IF
            no.of.results=P
              SEQ
                IF
                  display = 0
                    cont.in.out ! display.data;(P*size)::data
                  TRUE
                    SKIP
                no.of.results:=0
                IF
                  P=4
                    SEQ
                    SEQ i=0 FOR N
                     SEQ j=0 FOR N
                      SEQ
                        xi[i][j] :=fone
                        x[i][j] :=fone
                    vr2dfftl(x,xi,N)
                  TRUE
                   SKIP
                 process.finished:=TRUE
                 cont.in.filing ! B;process.finished
               TRUE
                SKIP
          filing.cont.in ? CASE
           display.data;size::data
            cont.in.out ! display.data;size::data
           w.terminate
            processing:=FALSE
           exe.terminate
            processing:=FALSE
           b;processing
             SKIP
           net.s;P;display
             SKIP


      :


PROC output.controller(CHAN OF grid app.in,
                    filing.cont.out,cont.in.out,
                    CHAN OF inter cont.out.sc)
   INT lo,e,N ,NN,Pn,k,no.of.results,convolution:
   VAL max IS 65600:
   [max]BYTE x:
```

```
[4200]REAL32 r.data:
INT any,time, result,len:
BYTE letter,length:
INT start,end,size,time.of.processing:
TIMER clock:
BOOL  continue,data.recieved:
BOOL continue,not.halt,processing,another,process.finished:
SEQ
 processing:=TRUE
 no.of.results:=0
 clock ? start
 WHILE  processing
  ALT
   filing.cont.out ? CASE
    display.data;size::x
     app.in ! display.data;size::x
    conv;convolution
     app.in ! conv;convolution
    real.data;Pn;size::r.data;lo;e
     app.in ! real.data;Pn;size::r.data;lo;e
    w.data;Pn;size::x;lo;e
     app.in ! w.data;Pn;size::x;lo;e
    b;another
     SKIP
    w.terminate
     SEQ
      clock ? end
      time.of.processing:=end-start
      VAL string IS "*c*ntime.of.processing:=":
      SEQ
       length:=BYTE (SIZE string)
       cont.out.sc ! sc.string;length::string
```

```
          cont.out.sc ! sc.int;time.of.processing
          cont.out.sc ! f.terminate
        processing:=FALSE
    exe.terminate
      SEQ
        processing:=FALSE
        VAL string IS "*c*nprogram deadlock":
        SEQ
          length:=BYTE (SIZE string)
          cont.out.sc ! sc.string;length::string
        cont.out.sc ! f.terminate
    cont.in.out ? CASE
      display.data;size::x
        app.in ! display.data;size::x
:
PROC hang.free(CHAN OF grid hang.free.filing,filing.hang.free)
-- This procedure is to terminate the program if deadlock occurs in the network
  INT time:
  TIMER clock:
  SEQ
    clock ? time
    ALT
      filing.hang.free? CASE
        w.terminate
          SKIP
      clock ? AFTER time PLUS 30000000
        hang.free.filing ! exe.terminate
:
PAR
  PAR
    input.controller (app.out,filing.cont.in,cont.in.out,cont.in.filing,cont.in.sc)
    output.controller (app.in,filing.cont.out,cont.in.out,cont.out.sc)
    screen.handler (screen,echo,cont.in.sc,cont.out.sc,filing.sc)
    hang.free( hang.free.filing,filing.hang.free)
    filing (keyboard,echo,to.filer,from.filer,
          filing.cont.out,filing.cont.in,
          hang.free.filing,filing.hang.free,
          cont.in.filing,filing.sc)
:
host (keyboard,screen,from.filer,to.filer, app.in,app.out,TRUE)
```