A Graph Theoretic Approach to Transputer Network Design For Computer Vision

By

Salim Omarouayache

Thesis submitted for
the degree of Doctor of Philosophy

City University
Department of Electrical and Electronic Engineering
Centre for Information Engineering

April 1995

Table of contents

Acknowledgements

I would like to express my gratitude to my supervisor Dr. T.J. Ellis for his guidance, support, cordial availability and patience.

Thanks are also extended to the students and staff of the Machine Vision Group in particular and the Department of Electrical and Electronic Engineering in general for providing the friendly atmosphere in which this work was carried out.

Financial support from the Algerian Ministry of Higher Education and the Kitchen Award are gratefully acknowledged.

Abstract

The work in this thesis is concerned with parallel architectures based on the Inmos transputer-type processors and parallelisation of some computer vision tasks chosen from low to high level.

The transputer is a microprocessor with a micro-programmed scheduler and four serial communication links. It directly supports parallel processing since several transputers can be connected through their links to co-operate on solving a problem. Also several processes can be run on the same transputer. A major issue in parallel processing is the communication overhead introduced by parallelising a given task. This overhead is not present in sequential processing and must be curbed if the implementation of a task on a parallel machine is to be successful. The interconnection network underlying the architecture of a parallel computer is therefore of the utmost importance.

Computer Vision consists of a hierarchy of tasks ranging from low-level operations dealing with large amounts of relatively simple data to high level operations handling increasingly complex structures. In this work a novel edge detector based on adaptive filtering and an edge detector operating on colour images are presented and implemented on a number of transputers. These parallel implementations together with implementations of vector quantisation, Fourier descriptors for shape discrimination, the Hough transform and the Maximum clique algorithm, offer a notable performance increase when compared with sequential implementations. However, every algorithm required the design of a specific network of transputers to take advantage of the parallelism and data dependencies inherent in each.

Consequently, attention is focused on the topology of interconnection networks. In particular, the communication requirements of computer vision algorithms as identified by the various computer vision tasks are analysed. These requirements together with graph theoretical considerations are then used to suggest a topology for large transputer networks. The latter is based on sub-graphs, with proven performance when used to implement interconnection networks, combined to form an architecture with improved performance. This architecture consists of a fixed structure supplemented with a dynamically reconfigured network. After describing this topology, a routing algorithm that conveys messages along shortest paths in the network is given and implemented. And finally, some practical issues in the use of transputers are considered and solutions proposed.

1. Single Instruction stream, Single Data stream (SISD)
2. Single Instruction stream, Multiple Data stream (SIMD)
3. Multiple Instruction stream, Multiple Data stream (MIMD)

Flynn's taxonomy, though restrictive in that it cannot classify some of today's computers/algorithms, serves as a guideline in the approach to new problems. Besides, to attain the high performance required the best of different system architectures is needed. Therefore, Flynn's taxonomy will probably be used in the future to classify sub-systems instead of complete computers.

Amongst the areas of research that require vast processing power is computer vision. Other areas include fluid dynamics, meteorology and numerical analysis.

The main focus of this work is on computer vision. It is believed that computer vision with its wide range of needs is a challenging area for parallel systems. In fact computer systems manufacturers often demonstrate the power of their machines on image processing applications. On the other hand, some problems in computer vision can only be solved -using current knowledge- with very fast computers. New methods and techniques that decrease the complexity will probably be discovered, but a large amount of processing is very likely to remain necessary.

The object of the work presented here is to study computer vision in the wake of parallel processing. This leads to the realisation that systems which offer good performance for one type of application may fail to do so for others. Commercially available hardware can compute image filtering tasks in real-time (25 frames per second). But, can this hardware search a large database using sophisticated data-structures for a best match? The answer is no. Thus, there seems to be a need for architectures and computer vision algorithms from various levels that map efficiently on them. At first glance the only solution seems to be a compromise between the models that have been used successfully for problems with fundamentally different needs. In other words, design factors such as grain of parallelism (size of atomic tasks), the ratio of inter-processor to processor-memory bandwidths, etc. are to be balanced. An important aspect in the design of such computers will be the unit Processing Element (PE). The latter should have an important bearing on the approach adopted. For example, intuitively a systolic array type of processor with very limited computing capability drawing its power from the regular beat of data, is not likely to be amenable to the solution of a combinatorial problem with non-deterministic spatial distribution of processing. For a start the data-structure representing the problem has

to be transformed into a set of homogenous and similar (and small) regions. This will have to be done on a powerful central computer and can be a complex problem.

Computer vision has traditionally been divided into a hierarchy of processing steps. The lower steps are named low-level vision and deal with image level operations. A range of operations (which is not so well defined) can be termed intermediate-level vision and involves grouping the low-level features extracted by the previous steps into more meaningful entities. The latter are operated on by processes categorised as high-level vision which are concerned the reasoning aspects of computer vision. The processing and communications requirements of parallel implementations of tasks from different levels will differ greatly. This is the main reason for the choice of the algorithms in Chapter 4, Chapter 5 and Chapter 6. Edge detection is clearly a low-level vision task. The Fourier descriptors and the Hough transform fall into the intermediate-level. Finally, the clique finding algorithm falls into the high-level category. The aim of the various implementations is two-fold. First, it will be shown that the communication needs are such that different algorithms require different topologies (even for a modest number of processing elements). Therefore, a flexible architecture is needed, especially for large networks. Second, improvements to the algorithms will be proposed.

The computer vision algorithms studied in this work are edge detection, vector quantisation, the Fourier descriptors, the Hough transform and the maximal clique. These algorithms are designed and implemented in parallel. First, a new edge detection operator is formulated based on the adaptive filtering method. Second, a modified Canny edge detector is applied to the three planes (R, G and B) of a colour image with corroboration between the different planes. Third, a vector quantisation algorithm is defined, where the coder is implemented on a network of processors. Then, a shape discrimination scheme based on the Fourier descriptors is formulated. The latter together with the Hough transform algorithm present examples of middle-level vision algorithms. Finally, the maximal clique finding algorithm is a representative of high level computer vision.

Graph theoretic methods have been shown to allow the expression of many search problems. Besides, many problems in computing reduce to enumerations or combinations; and most problems in combinatorics can be expressed in graph theoretical terms. Some of the seemingly intractable (without very involved calculus) problems of statistical mechanics have been solved using graph theoretical methods [Temperley]. More in line with this work, relational structures can be represented as graphs [Ballard]. Thus, matching objects in a scene to models in a database reduces,

given the right interpretation of nodes and arcs, to graph searching procedures. One such method involves forming an *association graph*[2] from the two relational structures representing the object in a scene and the model. Having formed the association graph the problem becomes that of finding its *cliques* i.e. sets of completely connected vertices. A clique that cannot be extended by adding a vertex to form a larger clique is a *maximal clique*. Therefore, good matches between object and model will correspond to large cliques. The parallel implementation of this procedure is the subject of section 6.3.

The Hough transform (section 6.2), and Fourier descriptors (section 6.1) are considered next. The former is implemented for both straight lines and generalised parametric curves. The complexity of the algorithm is analysed and a method is proposed for further performance increase. The latter is also implemented on a network of transputers and the use of the Hartley transform is proposed as a faster alternative for the Fourier transform. The notion of normalisation is also considered.

Problem solving using parallel processing requires, at the outset, a choice of approach. There are two main categories of parallel systems:

1. shared memory parallel computers,
2. and distributed memory parallel computers.

The work presented in this report concentrates on a sub-class of the distributed memory parallel computers that rely on the message passing paradigm.

The aim here is, therefore, to study the current state of affairs in network topologies; and to use graph theoretic concepts and practical considerations to present an architecture with 'good' characteristics. The latter are based on both formal metrics drawn from the theory of graphs and computational concepts. The view is taken in this work that graph theory is a powerful tool in the design of parallel computers and parallel algorithms. The connection with the topologies of networks of processors is intuitive. One can easily picture the PE's as vertices and the communication links as edges of a graph. Examples of the use of graph theory in the design of minimum latency reliable networks predate parallel computers.

[2]Please refer to section 6.3 for the relevant definitions.

The approach adopted here relies on making use of graph theoretical concepts and results, mainly the diameter (or maximum distance between any two nodes) but also the notion of dominators in directed graphs, etc. to reduce the inter-dependence of PE's that are not functionally related. Another important aspect of architectures is the ability to map useful partial-graphs that have been shown to implement certain algorithms efficiently e.g. trees, meshes, etc. Besides, the communication structure of a problem can vary within one level of processing as well as between levels. This suggests that reconfiguration should be considered. However, general reconfiguration routines require a great deal of synchronisation and thus introduce further sequential processing (Figure 1.1). Hence, the use of a fixed topology with 'good' characteristics as defined earlier and only a sub-set of links dedicated to reconfiguration is the method adopted.

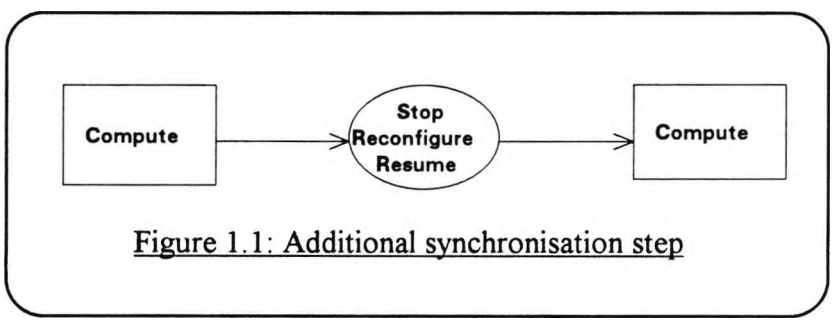

Figure 1.1: Additional synchronisation step

The advent of dedicated circuits to effect message routing represents a different approach to fast communication. However, these routers introduce delay and, in large networks where several are combined to implement full inter-connection between the processors in the system, there is no deterministic way to evaluate communication latency accurately.

In the next few paragraphs the structure of this report is introduced. This will include a short description of the content of each chapter.

Chapter 2 introduces computer vision and pinpoints the main differences in processing requirements between the different levels of its hierarchy of tasks. It describes selected tasks from low level image processing to the high level processes of knowledge representation.

Chapter 3 describes the parallel models, architectures and paradigms. It gives the old arguments against the use of parallelism and their rebuttals. Then, it describes some commercially available parallel computers and the rationale behind them.

Chapter 4 introduces two edge detection algorithms and their parallel implementation on transputer networks. First, an adaptive approach to edge detection is presented where the edge operator's kernel is determined from the image using a Widrow-Hoff filter [Widrow]. Then, a modified Canny edge detector [Canny] is adapted to colour images. The result of applying the operator to one of the planes (R, G or B) is corroborated by that of applying it to the others. Finally, both algorithms are implemented on specific transputer networks.

Chapter 5 presents the implementation of a parallel vector quantisation algorithm aimed at image compression. The implementation achieves very good speed up characteristics.

Chapter 6 presents the implementation of three different algorithms from computer vision. First, the Fourier descriptors technique is introduced and a fast algorithm based on substituting the fast Hartley transform for the fast Fourier transform is proposed. Two formulations of the Fourier descriptors are considered. One formulation has fallen into disuse because of the high frequency content of the resulting descriptors. An attempt at providing a solution is given. Second, the Hough transform technique is presented and a parallel implementation is described. The latter includes both the Hough transform for lines and its generalisation to arbitrary parametric curves [Ballard 2]. Last, the high level goal of matching models in a database to objects in images is considered. The associative graph technique based on finding the cliques is analysed in order to identify potential parallelisms. The problem of finding cliques in a graph is known to be *NP-complete*[3] i.e. there exists no deterministic algorithm to solve the problem in polynomial time (to the size to the input). However, there are in the literature many examples of simple heuristics used to improve the situation considerably [Bolles]. An algorithm based on block cluster analysis is presented, the main idea is to represent the association graph by its adjacency matrix and then use row and column permutations to bring the '1' entries close to the main diagonal. Also a parallel iterative version of the algorithm given in [Bolles] is presented. Finally, the transitive orientation of an undirected graph is shown to ease the problem of finding cliques [Liu]. However, not all graphs are transitively orientable; thus the use of transitive orientation is investigated as a simplifying procedure.

Chapter 7 deals with the design of a transputer architecture suitable for computer vision. The design criteria used to this effect build on graph theoretical results to find a

[3]Please refer to appendix B for the definitions of complexity theory.

topology that exhibits a reasonably small diameter, vertex and edge symmetry, etc. Then it turns to the possible schemes for reconfiguring the proposed architecture dynamically.

Chapter 8 presents a routing strategy for the architecture of Chapter 4 and presents tools for performance evaluation and other aspects of the implementation of algorithms on transputers. The programming language used throughout is Occam which is pared down and offers explicit support for parallelism and the message passing paradigm. Occam is the implementation of a subset of Communicating Sequential Processes (CSP) [Hoare]. CSP is a mathematical theory which views systems as a set of co-operating processes that can be defined formally; and manipulated to prove correctness, etc. Appendix A is dedicated to the transputer and Occam.

Chapter 9 is the conclusion; it summarises the main results of this work and considers extensions and future work.

CHAPTER II

2. Computer Vision

Computer Vision is the area of computing involved in automating the processes of visual perception. It includes image processing which deals with transforming, encoding and transmitting images. It also includes pattern classification which deals with separating or classifying features. Finally, computer vision includes higher level goals and techniques suitable for geometric and cognitive processing.

This chapter is intended as an introduction to the domain problems tackled, with special emphasis on the processing requirements of the different classes of algorithms. Computer Vision is introduced by groups of algorithms acting on different image representations. Figure 2.1 [Ballard] shows the relationship between the various representations that are of interest in computer vision.

Computer vision presents a hierarchy of tasks ranging from low-level perceptive processes to high-level cognitive processes. With regard to the processing requirements of the different levels of this hierarchy, it can be noted that low-level vision usually relies on relatively simple tasks handling large numbers of primitive data items (e.g. pixels), whereas high-level vision consists of complex tasks handling more evolved representations (e.g. relational structures).

An important factor in parallelising an application is load balancing, because one must ensure that no processor (or processors) remains idle while others are overworked. In chapters 4, 5 and 6 a number of computer vision tasks are considered and parallel algorithms are presented and implemented.

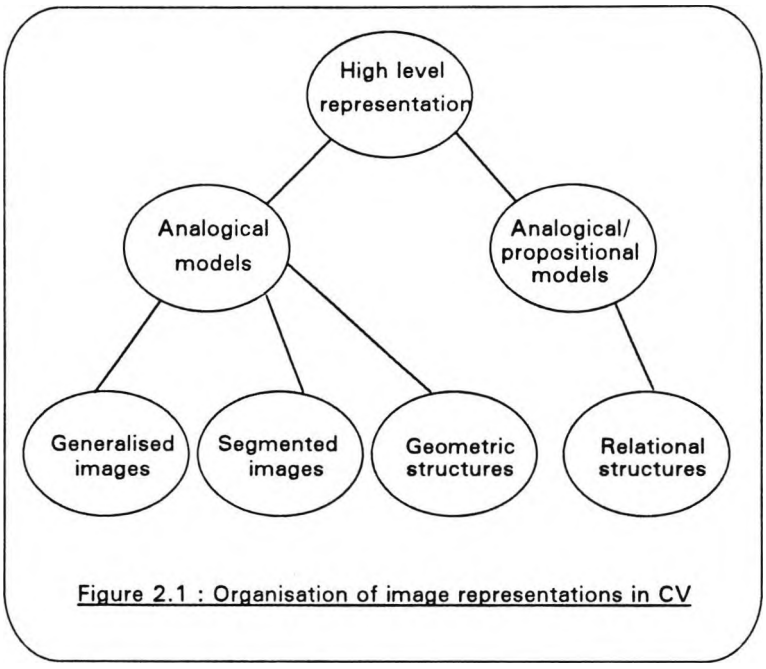

Figure 2.1 : Organisation of image representations in CV

Ballard and Brown categorise the representations of computer vision into four main groups as follows:

1. Iconic images,
2. Segmented images,
3. Geometric structures and
4. Relational structures.

Moreover, they identify a loose ordering between the different representations. Also, each category can be made up of several layers of representation. Such a view of computer vision is very helpful from a parallel processing view point, since representations within the same category will have similar storage and communication structures.

The input to a computer vision system is invariably a digitised image (or set of digitised images) produced by some piece of equipment that converts radiation e.g. X-rays, ultrasound, visible light etc. into an electronic signal which is then sampled. The most common radiation used is visible light; a camera e.g. CCD[1] produces a two-dimensional array of pixels which can be displayed or passed on to the computer vision system for processing. This array is an iconic representation of the scene of interest. Most processing at this level will produce another iconic image e.g. edge pixels. However, intrinsic properties of the scene can also be produced which give

[1] An array of Charge Coupled Devices have their capacitances modulated by the incoming light.

information about surface reflectance, orientation, depth of field etc. The major common property of these low-level processes is that data dependency is local i.e. to compute the output of the process at a particular location only a close neighbourhood around that location is needed. Therefore, these processes exhibit a high degree of low grain parallelism and are adequate for SIMD implementation (especially processor arrays).

Segmented images are formed from iconic images associating sets of pixels with objects. Processes at this stage can benefit from knowledge about the context of the scene in order to reduce computation time. Data dependency is no longer local, since regions (segments) of the image might span a large area. However, data can still be partitioned and processed on a parallel computer and then recombined. An important issue will in this case be the amount of effort required to reconstruct the full image; does it warrant the decomposition ?

Geometric representations are concerned with quantifying the notion of shape. These representations are used both for storing prior knowledge about the world and current visual input. Thus, the geometric structures can be used to measure the difference between two scenes concentrating on shape. The representation is a lot more compact than an iconic image; hence a notable reduction in the amount of data handled by algorithms is achieved.

Relational structures are an amalgam of representations used to effect high level goals. Inferences have to be made based on these structures which accommodate the notion of shape, relative position, and other concepts e.g. inside, outside, etc. Processes at this level are not considered to be good candidates for parallel processing. This is due to the fact that reasoning about a structure requires global knowledge, and subdividing either data or function might involve complications with regard to communication. The latter will either be prohibitive i.e. processors will spend most of their time communicating instead of performing computations, or non-deterministic in which case there is no means of evaluating the complexity of the parallel implementation. However, insight into the workings of the algorithms involved can sometimes show hidden parallelisms which allow for small but non-negligible speed-ups .

It is apparent from the ongoing analysis that the hierarchy inherent to computer vision introduces major differences in the algorithms that tackle the different levels. Therefore, parallel implementations will require different approaches based on the data dependencies and computation to communication ratios. Whether the approach relies

on data parallelism or functional parallelism the problem of load balancing must be addressed. For example, a low level computer vision task running on a processor array must ensure that some processors are not starved while others are working to or beyond their capacity[2]. Also, a task that has been sub-divided into functional sub-tasks implemented on a pipeline architecture must ensure that no task is significantly more time consuming than the others, since the upper bound of the throughput of a pipeline is (once all processors are computing) limited by the slowest sub-task.

In each section, established algorithms are described. Furthermore, an attempt is made at recognising the computing requirements and presenting previous attempts at solving the work load problem and implementing the algorithms efficiently.

2.1. Early processing

The algorithms described in this section act upon the generalised images introduced above. Early processing includes filtering, edge detection, range transforms, surface orientation, optical flow, etc. The operations of interest in this work are filtering and edge detection. Therefore, they are the subject of the remainder of this section.

2.1.1. Filtering

Filtering in image processing relates to the transformation of the grey levels in an image so as to enhance or de-enhance some features of interest. This is mainly an extension of filtering in (time) signal processing to two-dimensional signals (images); that is, the image is convolved with a kernel representing the impulse response of the operation desired.

The process of 2-dimensional convolution of an image is the action of comparing a reference kernel with a small neighbourhood at every pixel in the image[3]. The general formulation of such an operation is as follows:

$$F(i,j) = \sum_{k=-n}^{n} \sum_{l=-m}^{m} I(i+k, j+l).h(k,l)$$

---

[2]This situation can arise (in applications that use buffers) when some processors are able to service their local buffer several times before other processors service theirs once.

[3] A number of pixels around the boundary will be undefined in the output image.

where h( ) is an mxn kernel, I() is the MxN input image and the output image F() has dimensions (M-m+1)x(N-n+1). Note that m and n are assumed odd as is often the case. This due to the fact that many kernels are odd or even functions of two parameters; and the result is stored in the location corresponding to the central pixel. A filter is completely defined by the values of the kernel.

2.1.2. Low pass filtering

Low pass filtering is generally used to reduce speckle noise and isolated noise pixels. However, since edges of objects are high frequency transitions they are adversely affected. Therefore, the filter coefficients have to be chosen carefully and in an application dependent manner. A few filters have proved useful in many applications [Gonzalez][Ballard][Duda] e.g. neighbourhood averaging and the gaussian filter. The neighbourhood averaging operation defines its coefficients so as to replace the pixel at the centre by the average over the overlapped neighbourhood in the image. The gaussian filter kernel is defined as follows: $h(k,l) = e^{-\frac{k^2+l^2}{2\sigma^2}}$ assuming the same standard deviation is used in all directions. Note that because of the separability of the variables the gaussian operator can be applied through two 1-dimensional kernels[4], thus saving on computations while achieving similar results.

The major advantages of the gaussian operator besides this computational efficiency are:

- Its impulse response is gaussian, hence small span,
- it is simple to parameterise.

2.1.3. High pass filtering

High pass filtering is used to enhance the large local transitions in pixel values. This has the effect of sharpening the edges. The Laplacian filter is an approximation to the Laplacian of a function $\partial f^2/\partial x^2 + \partial f^2/\partial y^2$. The kernel is given by:

$$\frac{1}{8}\cdot\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

[4]Horizontally and vertically.

The Laplacian filter suffers from the lack of directional information and the fact that it relates to the second derivatives further enhances noise. Therefore, it is not very well suited for edge detection as a one stage process[5]. However, it is still quite useful as an edge enhancement technique. Also, it is often needed in conjunction with smoothing operators to reduce noise.

2.1.4. Edge detection

Edge detectors are an important part of many computer vision systems. They serve as a data reduction tools by simplifying data before further processing. For example, ([Binford]) constructing a geometric model to match against a database in industrial inspection involves locating edges before linking them to produce line and curve segments that are then combined to form the geometric model. Almost all vision systems use an edge detector as a front end (e.g. Acronym [Binford]). Line finders, whether they are based on edge following or the Hough transform, require data to be presented as an edge map.

Edges in an image can be associated with high gradient magnitude. This led Roberts to design the first edge detector as an approximation to the derivative of the two-dimensional function that is an image. The gradient magnitude d(x,y) and direction $\phi(x,y)$ are evaluated as follows [Ballard]:

$$d(x,y) = \sqrt{\Delta_1^2 + \Delta_2^2}$$
$$\Phi(x,y) = \tan^{-1}(\Delta_2 / \Delta_1)$$

where

$$\Delta_1 = f(x+n,y) - f(x,y)$$
$$\Delta_2 = f(x,y+n) - f(x,y)$$

Where *n* is a small integer defining the span of the operator. The span has to be large enough to accommodate small changes in the image but small enough to focus on local changes. The Roberts operator assumes by definition a step edge model. Also, it is sensitive to noise. To reduce the effect of noise various operators have been proposed. The Sobel edge detector, though based on the same principle achieves better

[5]The Laplacian opertor has been used as an adjunct to other operations to detect edges [Marr.]

performance by introducing local averaging. The kernels for the horizontal and vertical Sobel edge detector are:

$$\Delta_x = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \Delta_y = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The gradient magnitude g(x,y) and orientation θ(x,y) are computed as follows:

$$g(x,y) = \sqrt{\Delta_x^2 + \Delta_y^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{\Delta_y^2}{\Delta_x^2}\right)$$

where Δ_x^2 and Δ_y^2 are the values of the convolution of the respective kernel (Δ) with the image at location (x,y). Note that the operation is not defined for border pixels.

The Sobel operator compares favourably with the Roberts and other application of the gradient operator on the input image; however, it is still sensitive to noise producing a loss in detection accuracy. Kittler [Rosenfeld] proposed an iterated application of the operator as a means of improving accuracy. See [Kittler] for details.

Marr and Hildreth [Marr] proposed an operator based on the Laplacian of a gaussian filter. Edges are detected as zero-crossings in the output of the image convolution with the Laplacian of a gaussian.

The gaussian function has very interesting properties. First, as stated above, application of an n-dimensional gaussian can be performed by computing n 1-dimensional gaussian; thus, reducing computations. Second, the smooth shape of the gaussian means that it has good frequency characteristics and minimises the pass-band to stop-band transition problems (Gibbs effect).

Canny [Canny] proposed a set of criteria for optimal edge detection for a step edge model. The criteria are quite intuitive and can be stated as follows (1-dimensional case):

1. The edge detector must achieve low probabilities of false alarms and failures to mark real edge points i.e. large signal-to-noise ratio.
2. Good localisation i.e. the edges reported should be as close as possible to the real edges.
3. Single response i.e. an edge in the image should be reported only once.

After enunciating the set of desirable conditions Canny set about formulating them mathematically. The signal-to-noise ratio maximisation is given in the case of a finite impulse response [-W/2,W/2] filter (f(x)) applied to an edge located at x=0 (G(x)) and bathed in white noise (with mean-squared amplitude n_0)[6]:

$$\text{SNR} = \frac{\left| \int_{-W/2}^{+W/2} G(-x) f(x)\, dx \right|}{n_0 \sqrt{\int_{-W/2}^{W/2} f^2(x)\, dx}}$$

To achieve accurate localisation, the root-mean-squared distance of the marked edge from the centre of the real edge is minimised. This can be achieved by noting that the real edge is assumed to be located at x=0 but because of noise it will be marked at x_0. Therefore, localisation can be improved by minimising the standard deviation of x_0. A simple analysis of the equations involved led Canny to propose the following measure:

$$\text{Localisation} = \frac{\left| \int_{-W/2}^{W/2} G'(-x) f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W/2}^{W/2} f'^2(x)\, dx}}$$

The above equation is an approximation to the reciprocal of the standard deviation of x_0. Hence, a simultaneous application of the first two criteria proposed is equivalent to the maximisation of the product of the two equations i.e.

[6]Edges are marked at the local maxima in the response of the filter f(x).

$$\frac{\left|\int_{-W/2}^{+W/2} G(-x)f(x)dx\right|}{n_0\sqrt{\int_{-W/2}^{W/2} f^2(x)dx}} \cdot \frac{\left|\int_{-W/2}^{W/2} G'(-x)f'(x)dx\right|}{n_0\sqrt{\int_{-W/2}^{W/2} f'^2(x)dx}}$$

Using the Schwarz inequality for integrals it can be seen that the function which maximises the product is f(x)=G(-x) $x \in [-W/2, W/2]$. This is hardly surprising since f(x) becomes the matched filter; and the matched filter's performance represents an upper bound to the improvement in signal-to-noise ratio through linear filtering.

The third criterion proposed by Canny is required because of the frequency characteristic of the step edge. Such a filter has a high bandwidth and outputs many maxima as a response to a noisy edge. Therefore, a further constraint has to be formulated. A result due to Rice [Rice] and reported in [Canny] states that the average distance between zero-crossings of the response of a function to gaussian noise is:

$$x_{ave} = \pi \cdot \left(\frac{R(0)}{R'(0)}\right)^{1/2}$$

where $R(\tau)$ and $R'(\tau)$ are the auto-correlation functions of the function and its first derivative respectively. The distance between adjacent maxima in the noise response of the function is $2*x_{ave}$. If this distance is set to a fraction k of the function's width W, the expected number of maxima due to noise will be $2/k$.

Having formulated the constraints for edge detection Canny solved the optimisation problem numerically. He then proposed the derivative of a gaussian as an approximation to the optimal edge detector. Hysteresis thresholding was then applied to the output in order to grow back noisy breaks in contours. This amounts to edge linking.

The Canny edge detector is amongst the most computationally intensive algorithms considered in this section. However, its data dependency is local and it is well suited for fine-grain SIMD implementation. Therefore, custom VLSI hardware is in principle the best approach, because the performance of the few instructions used can be optimised and the architecture of single nodes can be devised according to the data paths in the algorithm. Ruff [Ruff] proposed a pipelined architecture achieving video rate throughput. The process outputs 8-bit edge strengths, 8-bit directions and 8-bit

sub-pixel position (1/50th). The implementation is based on noting that the Canny operator can be thought of as a three stage process Figure 2.2 :

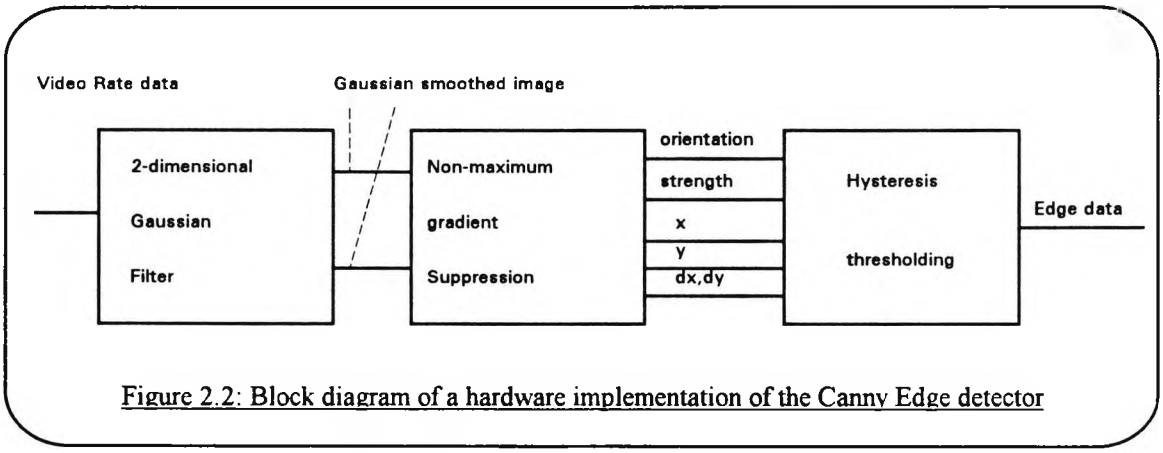

Figure 2.2: Block diagram of a hardware implementation of the Canny Edge detector

Ruff's work is a clear example of the approach that can provide real-time performance for the algorithms of low level vision. However, reasonable speedups can be achieved using a general purpose multi-computer architecture (e.g. transputer network) as shown in Chapter 5 which presents an adaptive filter approach to edge detection. Also, Chapter 6 describes an implementation of the Canny edge detector for colour images. The two appendices present attempts at improving the performance of edge detection both in output and performance.

Other authors have studied the Canny optimality criteria and proposed alternative methods based on infinite impulse response (IIR) formulations [Deriche]. Another parallel diffusion based implementation was given by Belhaire *et al* [Belhaire]. Deriche was concerned with the choice of the derivative of a gaussian as an approximation to the ideal filter Belhaire *et al* aimed at the hardware implementation of a parallel optimal filter.

Various methods and approaches have been proposed for edge detection. Very often the goal is to improve the quality of the output in situations where the models given above (step edge, etc.) are inadequate [Gupta]. Frei and Chen [Frei] proposed a method to detect edges based on template matching. They derived a set of 9 masks (3x3) which constitute a set of orthogonal basis functions. Each basis vector corresponds to a local interpretation of the edge situation for a patch (3x3). Convolution of the input image with the 9 masks is equivalent to the projection of the image on each of the sub-spaces generated by the basis vectors. The projection of highest magnitude is the best approximation to the local edge.

Gupta and Wintz [Gupta] present an algorithm based on hypothesis testing to locate boundaries in grey level images. The image is sub-divided into small patches (e.g. 2x2 arrays) then the first and second order statistics of adjacent subsets are compared. Similar patches are combined to form blobs. After the entire image has gone through this process, a segmented image is obtained. The boundaries between blobs, then, represent the boundaries between objects in the original image. The impetus behind the approach proposed by Gupta and Wintz was provided by the fact that finding edges in images, based on gradient methods, (pre-1975 [Pingle], [Rosenfeld].) did not offer good-enough noise rejection to validate boundary finding. Although edge detectors have since improved greatly, the mathematical models adopted for the design of filters are still almost invariably over-simplified (step, ridge, roof and white gaussian noise). Therefore, an approach similar to the one proposed should be of value when the assumptions about noise statistics are ostensibly inadequate (e.g. boundaries between textured regions).

The algorithm described in the previous paragraph does not completely fit in with the rest of this section. This is due to the lack of knowledge about the way blob combination is to proceed. However, the operations of blob comparison and evaluation of statistics are still local and within the framework of multi-processors with powerful PE's like transputers, SIMD seems a reasonable approach at the early stages of execution. Combining boundaries accumulated on separate processors would, however require communication to link the blobs.

2.2. Segmented images

Segmentation in computer vision originated from work in psychology concerned with tendencies in human perception to group shapes and features in the visual field. Features like proximity, similarity and continuity seem to be extensively used in human visual perception [Kohler]. Gibson and Gregory [Gibson] postulate that the grouping of features together with object/background discrimination[7] organise the scene into meaningful parts as a step towards image understanding.

Segmented images differ in many ways from the input (generally digitised) images. First, domain dependent aspects start playing an important role in the algorithms for segmented images. Second, data structures have to be designed for the representation

[7]Through edge detection, thresholding, etc.

of segments[8], in other words the 2-dimensional array representing an image is no longer appropriate for storing perceptual features. Last, the transformations involved in generating a segmented image require more elaborate computations that are likely to involve global knowledge about the scene; therefore, global communication will be required in a prospective parallel implementation.

In this section segmentation algorithms are introduced and separated into two main categories. First, the algorithms that perform segmentation based on boundaries between objects are described briefly. Then, region based segmentation is described.

2.2.1. Boundary based segmentation

Boundaries between objects in an image play an important role in the scene understanding process. Many different approaches have been proposed for their detection. The techniques advocated are based on a wide range of mathematical formulations of the problem at hand. Dynamic programming, the Hough transform, divide and conquer are but a few of the methods used in the literature. The algorithm described in the previous section achieves boundary detection through yet another technique based on the statistics of blobs accumulated in the image.

Dynamic programming is a recursive computational procedure used to solve optimisation problems [Liu]. It is particularly powerful in solving the decision problems of multistage processes. A multistage process is characterised by a set of state variables that fully describe the status of the process at any given stage. A decision has to be taken at each stage from a possibly infinite number of actions. For a problem to be amenable to dynamic programming, an objective function has to be specified. A decision taken at a particular stage affects the value of the objective function and the state variables of the next stage. Therefore, a policy is required for choosing the appropriate actions. Such a policy is optimal if it optimises the objective function. Optimality is the basic principle behind dynamic programming. It can be stated as follows [Liu]:

> *An optimal policy is one where, whatever the initial state of the process and the initial decision, the remaining decisions must constitute an optimal policy with regard to*

[8]Usually boundaries or regions in a scene.

the new state of the process resulting from the first decision.

The above definition results in the recursive nature of dynamic programming. In other words, since only the decisions that affect the subsequent stages are to be considered, a policy can start searching for the optimal decision from the last stage and then work backwards.

A boundary detection procedure can be so formulated that it encompasses the notion of best boundary (an objective function) [Ballard]. The input to such a procedure is an edge map as produced by the algorithms mentioned in section [2.1.4].

Ballard [Ballard 3] associates "goodness" of a boundary with a high cumulative gradient amongst connected edge pixels and low cumulative curvature. The following equation implements this objective function [Ballard].

$$h(x_1,\ldots,x_n) = \sum_{k=1}^{n} g(x_k) + \alpha \sum_{k=1}^{n-1} q(x_k, x_{k+1})$$

where α is negative, $g(x_k)$ is the gradient magnitude at kth point and $q(x_k,x_{k+1})$ is the difference between the gradient angle at the two successive points. Maximising h(...) constitutes following a set of disconnected edge pixels to form the most likely connected boundary containing a subset of edge pixels (according to the assumption built into the objective function). A recursion equation can thus be defined:

$$f_0(x_1) \equiv 0$$
$$f_k(x_{k+1}) = \max [\, g(x_k) + \alpha\, q(x_k + x_{k+1}) + f_{k-1}(x_k)$$

One of the major advantages of dynamic programming used as above for edge following is that a parallel implementation is feasible if many starting points are considered. A large set edge pixels is partitioned and each subset is processed on a separate processor and then the boundary segments are combined.

Heuristic search is a method used to reduce the computational complexity of graph searching. The object is to generate a path in a given weighted graph between node A and node B with minimum cumulative weight. Nilsson [Nilsson] formulated the optimisation problem by estimating the cost of travelling from A to an intermediate node I and the cost of traversing the path from node I to node B. The sum of these two costs is then the evaluation function which guides the heuristic search. The latter can produce reasonable (though sub-optimal) results fast when exhaustive search is

impractical. If more constraints are satisfied, the method always produces the minimum cost path.

Heuristic search for edge following was first proposed by Martelli [Ballard]. The method constructs a weighted graph from the output of an edge detector. The gradient angles provided by the edge operator are taken to be the nodes of a graph and weights corresponding to the gradient magnitudes are associated with these nodes. An arc is added to the graph if gradient angles are appropriately aligned. Then Nilsson's method is used to produce candidate paths that will correspond to boundaries in the original image. Valid evaluation functions are essential for the success of this method. Authors have proposed a mix of context dependent and general rules as evaluation functions [Ashkar][Lester][Ballard]. Heuristic search has proved quite powerful and compares favourably with dynamic programming [Martelli]. However, it does not present the inherent parallelism apparent from the dynamic programming formulation.

When an object of known shape is to be located in an image, a generalisation of the Hough transform [Ballard 2] is an efficient approach. The latter relies on an accumulator array, in which peaks are formed by counting the edge pixels supporting the presence of the shape in the input image. The accumulator array is a set of possible locations for a reference point fixed when the shape was first parameterised. Section 6.2 presents a transputer implementation of a parallel Hough transform. Therefore, a more complete analysis of the Hough transform is deferred.

A number of methods have been proposed for the manipulation of boundaries (not always linked to segmentation) to provide representations appropriate for higher level processing. Chain encoding [Freeman], Medial axis transform [Persoon] and Fourier descriptors [Persoon][Wallace] are all methods that apply satisfactorily to particular situations. In particular, the Fourier descriptor approach is presented together with a transputer implementation and the use of the Hartley transform in section 6.1. Wu and co-workers [Wu] proposed a parallel implementation of boundary manipulation algorithms based on chain codes and crack codes.

2.2.2. Region based segmentation

Region segmentation can be achieved through thresholding. A simple approach could be implemented by forming the histogram of grey levels in the image and then choosing thresholds to form regions. Obviously, the choice of thresholds is critical. Chow and

Kaneko [Gonzalez] proposed a method for optimal threshold selection for the two regions case.

Region based segmentation can be viewed as the problem of finding a partition of the input image into non-overlapping regions. One such method is region growing. The simplest region growing technique relies on properties of local groups of pixels. The process starts from a set of pixels and grows regions by appending to each point in the original set those pixels in its neighbourhood that have similar properties. Besides, the difficulty in defining adequate properties, this simple method suffers from its dependence on the choice of the original pixel set and ambiguities due to quantifying the properties.

Split and merge is another approach to region growing. The basic algorithm relies on the homogeneity of individual regions and the non-overlap between regions. If a particular region does not satisfy the condition then it is split into two regions. When there is an overlap between two regions, they are merged. Horowitz and Pavlidis [Horowitz] presented an algorithm to implement the split and merge operations towards segmentation using region growing. Other algorithms were proposed for boundary melting [Brice] which can be made more descriptive through the use of graph-oriented region structures [Ballard].

A class of techniques that can be used to perform region based segmentation are the so-called clustering and unsupervised learning algorithms [Duda]. These algorithms are very prominent in pattern recognition. The methods span a wide range of formulations and assumptions about the probability densities that describe the data. One algorithm that is typical of this class is the Isodata algorithm used for clustering. It starts with initial estimates of the means of clusters (regions in the case of image segmentation) and evaluates the variability of the regions formed by associating samples with a cluster according to a distance measure[9]. The so-formed clusters are split and merged across the dimension of highest within-cluster variance and lowest inter-cluster variance respectively. At each iteration the cluster means (or centres) are updated. This can be achieved through the K-means algorithm [Duda]. The K-means algorithm converges to a local minimum on the error surface and is usually used as an adjunct method to other algorithms. An implementation of such a scheme forms part of the Khoros system [Khoros] and was used for the segmentation of CAT scans in order to

[9]In this context the Mahalanobis distance [Duda] is often used because of its weighting of dimensions according to the directional variances.

detect brain tumors [Omar]. Coleman and Andrews [Coleman] use the K-means algorithm together with between cluster and within cluster scatter measures to implement segmentation through clustering.

2.2.3. Texture

Texture is an important feature in image processing. Often characterising texture in a scene is the goal of the application at hand. Moreover, texture can be used in image segmentation. A set of filters based on "texture energy" were introduced in [Laws]. Laws defines texture energy as *the amount of variation within a filtered image window*. Therefore, energy measures depend on the filter used and the method used to quantify the variations. A texture identification procedure based on this concept was presented in [Laws].

Mitchell *et al* [Mitchell] propose a max-min measure for texture analysis. Their method involves the relative frequency of local extrema in the grey level image as the principal measure. The simplicity of the method offers considerable reductions in processing time when compared with previous techniques. Davis, Johns and Aggarwal [Davis] present a different approach based on co-occurrence matrices and discuss features derived from them.

2.3. Geometric structures

Algorithms that operate on geometric structures produce shape descriptions of boundaries. Segmentation produces blobs with no explicit characteristics. The latter are synthesised, at this stage in a computer vision system, to produce more compact data structures amenable to the higher level goal of image understanding.

Polylines can represent curves by a succession of line segments. To arrive at a desired accuracy of representation an estimate of a good match has to be devised. One such method was proposed by Horowitz and Pavlidis [Ballard] in the case where the number of segments is known . It consists of locating corner points and splitting and merging line segments in a digitised image according to the degree of fit to a straight line . Several approaches and variations can be found in the literature [Duda]. They rely on splitting line segments, merging line segments or a combination of the two operations.

Chain codes were first introduced by Freeman [Freeman]. A chain code representation of a curve can be derived from its pixels by storing the direction of the next pixel along the curve as it is traversed (generally in counter-clockwise direction). Given that the image is digitised there are a limited number of directions for the next pixel. Therefore, the representation afforded by chain codes is quite compact and thus computationally efficient. Besides, chain codes offer simple methods for calculations of some parameters of closed curves e.g. perimeter and area.

Fourier descriptors represent the boundary of a region as a periodic function. The basic idea [Persoon]. is to store the curve as the coefficients of the Fourier series expansion of the 1-dimensional array of samples taken from the boundary. The main advantage of this method is that for a 'good' functional representation of the boundary samples, the Fourier representation will have very few coefficients. Section 6.1 will present further analysis of this method and a parallel implementation.

Another approximation to curves is offered by interpolative methods. Several polynomial interpolants are used in computer graphics and present good analytic properties making them easy to manipulate in image processing. Besides, polynomial interpolations yield aesthetically good curves and can approximate many natural shapes. B-splines are a concatenation of polynomial curves. The most frequently used polynomials are cubic since they are the lowest order polynomials that can represent concave shapes (they contain points of inflection).

Alternative methods for boundary representation are y-axis, quad-trees, and the medial axis transform [Ballard]. These methods together with the ones described above have advantages and disadvantages that warrant the choice of a particular method for a given application. In this work the choice of methods for further analysis was based on prospective performance improvement that can be achieved by parallel processing.

Three dimensional structures and algorithms can indeed benefit from an increase in performance. Many of the techniques used in the manipulation of 3-D objects rely on features and evidence accumulated from a 2-dimensional image. Then, one enters a level in the hierarchy of computer vision systems where the paradigms are those of high level vision (described next). One exception is, perhaps, the recovery of depth from an image e.g. from a single view point using vanishing points [Tai][Brillault].

2.4. Relational structures

This section presents the different aspects and tools of image understanding. These can be subdivided into major topics as follows [Ballard]:

1. Knowledge representation, and control
2. Matching,
3. Inference,
4. Planning.

2.4.1. Knowledge representation

Computer vision systems need a representation of the world. The purpose of this representation is to guide the different aspects of processing and reduce the amount of computation needed to infer world descriptions from the incomplete models of edges and other perceptual features.

For example, understanding or detecting and locating objects in the 3-dimensional world from a single 2-dimensional (or a few) view involves recovering depth of field information and approximating 3-d direction of object boundaries. One useful concept for this task is the 'Vanishing point' (VP). The idea stems from the field of projective geometry and can be stated as follows: Under perspective projection parallel lines in 3-d space form lines in the 2-dimensional projection space (the image) that intersect at a point VP. Several examples of the use of this technique to build perceptual groupings are present in the literature e.g. [Lowe], [Brillault].

Another important aspect of knowledge representation is the combination of perceived features (e.g. edges, texture, colour, etc.) In other words, there is a need for data models that can incorporate various kinds of information. The implementation of these data models should simplify access to areas of knowledge, e.g. through data abstraction, and allow for both the top-down and bottom-up processing. In this respect parallel processing (especially the MIMD model) can provide a means to achieve high performance through data and functional partitioning, and help reduce complexity through viewing a vision system as a set of co-operating processes[10].

---

[10]Processes would encompass knowledge about parts of the problem and connections (or channels) would represent dependency and (or) precedence.

Communicating Sequential Processes [Hoare] offers a framework where correctness and conformity to specifications can be proved.

Semantic networks present a very potent method for scene representation. A set of objects and their relationships are represented as a graph structure consisting of nodes and labelled arcs. Besides, the latter can have a value defining a particular characteristic of the relation.

2.4.2. Matching

Computer vision systems hold several representations of visual inputs and previous knowledge. These representations have to be integrated in order to achieve, for example, recognition. Therefore, structures derived from inputs have to be matched with internal representations. Hence, matching consists of an interpretation of input data associating different representations.
Graph theoretical algorithms are used in matching relational structures. Attempts at parallelising such an algorithm (clique finding) are presented in section 6.3.

2.4.3. Inference

Inference is the process of deducing new facts from a set of known facts. The most widely studied inference system is First order predicate logic. However, it is widely accepted that first order predicate logic does not address some important features of the reasoning performed by human beings. Therefore, workers have sought extensions to inference to improve the performance of computer reasoning. Production systems, relaxation labelling and active knowledge are such extended inference systems. Such systems tackle the problems of knowledge representation and implementation issues. They invariably involve search procedures operating on large databases. Workers in the field have concentrated a great deal of effort on reducing the search space by involving heuristics, etc. The emerging techniques of Distributed Artificial Intelligence (DAI) are directly amenable to multi-processor implementations.

2.4.4. Planning

Planning has traditionally dealt with robots performing actions in the real world. It involves decision making to derive the sequence of processing steps that will lead from a starting situation to a goal in an optimal fashion. In a computer vision system a planning task can be used to direct both the goal seeking aspects of the problem and

the data gathering stages. Parallel implementation could be beneficial if the planning task is communications and topology aware; thus, instantiating tasks and directing data to the appropriate resources.

2.5. Summary

This chapter presented an overview of the algorithms of computer vision. Although the emphasis was not put on parallelisation, it is believed to show that this extensive set of techniques and methods present a wide range of computational requirements. The latter form the basic justification for seeking a suitable parallel architecture (subject of chapter 7).

CHAPTER III

3. Parallel processing

From the early 60's many arguments against the use of parallelism were introduced. Most of these arguments were context and time dependent and were only valid within their respective hypothesis frameworks. Therefore, they failed to have a global perspective and are now easily refuted. In the following few sections these arguments and their rebuttals are given, then parallel machines are introduced and classified.

3.1. Arguments against the use of parallel machines

Amongst the researchers who did not believe in the potential merits of high-level parallelism were Grosh, Minsky and Amdahl [Quinn].

Grosh argued that the speed of computers is proportional to the square of their cost. Therefore, if you are looking for a faster computer, you are better off purchasing one large computer than two less performant machines and connecting them.

Minsky for his part believed that the speedup achievable by a parallel computer increases as the logarithm of the number of processing elements, therefore making large scale parallelism unproductive.

One of the most potent arguments against the future of parallel computers was Amdahl's law. The latter states that a small number of sequential operations can limit the speedup of a parallel algorithm. Let f be the fraction of operations that must be

performed sequentially ($0 < f < 1$). Then the maximum speedup S achievable by a parallel computer with n processors is

$$S \leq \frac{1}{f + (1-f)/n}$$

It can be seen from the above that a small portion of sequential operations can significantly limit the speedup achievable by a parallel computer.

Other arguments against large scale parallelism include software inertia[1] and the fact that most commercially available super-computers (up to the early eighties) were vector processors with appropriate vectorising compilers.

3.2. Rebuttal for the arguments against parallelism

Grosh's law suffers essentially from the limited performance of the current fastest computer. Therefore, it cannot hold true asymptotically. This leaves but one potential alternative for performance increase: **using more than one processor**.

Experimental results show that the speedup achievable in a parallel implementation depends on many factors:

- The architecture of the parallel computer (inter-connection network, link bandwidth, etc.)
- The particular algorithm (inherent parallelisms, nature of the communication between independent modules, etc.)

Therefore, Minsky's conjecture can be refuted on the grounds of experimental results. Some algorithms presented in this work (especially those of early vision) exhibit quasi-linear speedups, though for a modest number of processors.

As far as Amdahl's law is concerned there are some algorithms with very few sequential operations e.g. convolution of an image with a small kernel. However, the law can be used to assess the adequacy of algorithms as candidates for parallelisation.

---

[1] Billions of dollars have been spent in the development of Fortran software.

Limiting the means of achieving speedup in computers to vector processors excludes a large number of important problems. Besides, current super-computers e.g. Cray-3 contain several pipelined vector processors designed to work in parallel. It is reasonable to assume that parallel processors will allow for new and more challenging problems to be tackled; thus, the programmers of the future are likely to be involved in solving computationally intensive problems on the new architectures.

3.3. Models and paradigms of parallel computation (taxonomies)

In the Von Neuman model a single, possibly very powerful, central processing unit is connected to a single random access memory which stores both programs and data. The CPU and RAM communicate in a serial fashion through a narrow conduit termed 'the Von Neuman bottleneck' [Lipovski].

A paradigm is a set of architectures based on the same principles. The Von Neuman paradigm contains virtually all multi-purpose computers. The fundamental features of this paradigm are as follows: a controller, data operator, memory and input-output are sequentially programmed in a fetch-decode-execute cycle.

3.3.1. Flynn's taxonomy

Parallel architectures can be subdivided into two main categories based on data stream and instruction stream [Flynn]. An instruction stream is a sequence of instructions performed by a computer; and a data stream is the sequence of data on which the instruction stream operates. Flynn [Flynn] categorises an architecture by the multiplicity of hardware used to manipulate instruction and data streams. Multiplicity is the maximum possible number of simultaneous operations or operands at the same stage of execution at the most constrained component of the organisation. Therefore, four classes of systems can be distinguished[2].

The single instruction stream, single data stream (SISD) category contains most serial computers; although instructions can be pipelined, only one instruction is at any given stage of its execution e.g. fetch, decode, etc. This is primarily due to the fact that only one control unit directs the flow of instructions into the computer.

---

[2]Although only three are covered here since Multiple Instruction stream Single Data stream (MISD) is of no practical consequence.

In the single instruction stream, multiple data stream (SIMD) category a single instruction stream is executed by a number of processing units each capable of fetching and manipulating its own data. Therefore, a number of processors apply the same instruction to different data, at any one time.

The multiple instruction stream multiple data stream (MIMD) category contains multiprocessor systems that execute multiple instruction streams and manipulate multiple data streams. In other words, processors co-operate to solve a problem by solving components of the partition of the problem into sub-problems operating on members of the partitioned initial data.

3.3.2. Händler's taxonomy

Another classification method was proposed by Händler [Händler]. The method is based on a notation for the expression of parallelism and pipelining occurring at different levels of a computer. First, the Processor Control Unit (PCU) is described by two numbers; namely, P the number of individual PCUs and P' the number of PCUs that can be pipelined. Second, the Arithmetic and Logic Unit (ALU) is represented by two numbers: A the number of ALUs and A' the number of ALUs that can be pipelined. Third, the Bit Level Circuit (BLC) is defined by a further pair of numbers B and B'. B is the word length of individual ALUs and B' the number of pipeline sections in the ALUs. Therefore, a computer system is described by a triplet T as follows:

$$T(\text{computer}) = < PxP', AxA', BxB' >$$

This representation is more binding than Flynn's taxonomy because it quantifies the levels of parallelism that a given system can handle. However, it fails to describe the general MIMD model insofar as the function intended by the system designer can affect P, P', A, and A'. For example, the 2-dimensional transputer (T800) array (Figure 3.1) can be described as: <16x1,16x16, 32x1> or <16x1,16x1,32x1>, etc. depending on the functionality of each transputer

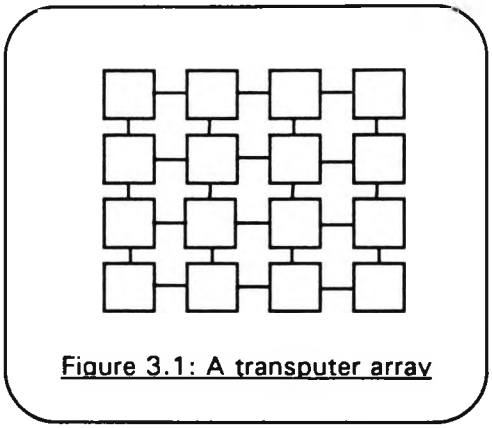

Figure 3.1: A transputer array

However, the notation is tight in that it explicitly represents the computable threads and the numbers to the left of 'x' give the number of nodes and the control structure.

3.3.3. Feng's taxonomy

Feng [Feng] suggested the use of the word length of the processing units (n) and the bit slice length m (the product of the number of pipelines and their depth) to classify systems. In his taxonomy Feng introduces 4 categories:

- A system is Word Serial, Bit Serial (WSBS) if $n = 1$ and $m = 1$.
- A system is Word Parallel, Bit Serial (WPBS) if $n = 1$ and $m > 1$.
- A system is Word Serial, Bit Parallel (WSBP) if $n > 1$ and $m = 1$
- And finally a system is Word Parallel, Bit Parallel (WPBP) if $n > 1$ and $m > 1$.

3.3.4. Skillicorn's taxonomy

Skillicorn [Skillicorn] introduced the idea of modelling the inter-connection networks that may exist within a system. Such networks include the processor to memory, processor to ALU, and processor to processor subsystems. Therefore, the system is characterised by the following variables:

- The number of instruction processors (I),
- the number m of instruction memories (M),
- the I to M network,
- the number of ALUs (D),
- D to data memory network,
- I to D network,
- and D to D network.

This system is very flexible and, depending on the interprocess notation, is capable of representing most current systems. However, it is a little cumbersome, and is probably best used in combination with Flynn's system (where only features that Flynn's taxonomy cannot differentiate are specified).

3.4. Different approaches to parallel computer design

Exploiting the parallelism inherent in different fields of application is a daunting task. As mentioned earlier, the structure of processing and communication dictates different and often incompatible architectures for various classes of algorithm.

Before considering purpose built parallel computers it is worth mentioning software environments (e.g. Alice, Parallel Virtual Machine (PVM), etc.) For example, PVM [Beguelin] is a public domain system that enables a collection of heterogeneous computer systems to be used as a coherent and flexible concurrent computation resource. The individual machines may be multiprocessors, vector supercomputers, specialised machines or scalar workstations, that may be interconnected by a variety of networks. Support software executes on each machine and presents a unified computational environment for concurrent applications.

This work is concerned with the MIMD model of parallel processing. However, there are other models that offer high general purpose performance e.g. processor arrays (ICL DAP) and vector supercomputers (CRAY 1). Vector supercomputers in particular have been very successful in the past in various application domains. However, even Cray Research Inc. (the Cray computer makers) have moved to designing massively parallel machines [Köninger]. The aim is to produce a heterogeneous environment of vector, scalar and parallel systems.

Before dwelling on the large number of possible approaches to the design of control structures for parallel computers, an initial choice concerning the Processing Elements (PE) has to be made. The two alternatives available are:

1. Connect together a relatively small number of very powerful PEs.
2. Use a large number of simple PEs.

The first approach shall be termed 'herd of elephants' and the second 'army of ants' [Quinn]. If the sequential fraction of a computation is greater than the ratio of processing powers (of the PEs of the two cases above PE_2/PE_1), then a single PE from

the first approach will offer better performance than the whole system of the second. This is because the inherently sequential part of the process as run on a single processor of the herd-of-elephants machine will be faster than the overall speedup achievable by the whole army-of-ants machine.

The approach adopted defines the hardware and software support required from the underlying machine. The aim of this section is to give a brief overview of the currently popular techniques. First, the army-of-ants approach is introduced through a commercially available system; then a few examples of the herd-of-elephants approach are given.

Many commercially available parallel computer systems implement massive data-parallelism as the means to achieve speed-up (ICL's Distributed Array Processor, Goodyear's Massively Parallel Processor, etc.) Thinking Machines Corporation's Connection Machine (CM) is characteristic of the army-of-ants approach to parallel computers design, and is described next.

3.4.1. The army-of-ants approach (the Connection Machine)

The Connection machine[3] (CM) [Tucker] is a data-parallel system which integrates hardware and software. It is controlled from up to four front-end computers which provide the development and execution environments. CM consists of 65536 processing elements and millions of virtual processing elements through its virtual processor mechanism. Processors are connected through a general purpose reconfigurable communication network. Therefore, CM contains all the software and hardware modules for the design and implementation of data-parallel algorithms.

The front-end computer systems connect through a 4x4 cross-point switch to four sequencers. Each sequencer controls up to 16384 processors. Moreover, a high performance data-parallel input/output (I/O) system connects processors to peripheral mass storage and graphic display devices. One of the major advantages of CM, besides the high performances achieved, is that system software is based on the operating system of the front-end computer (DEC VAX or Symbolics[4]). Thus, programmers can take advantage of the power of CM without the need to fully understand the underlying structure of the machine.

---

[3]The description of CM applies to both CM1 and CM2 unless stated otherwise in the text.

[4]With minimal visible software extensions [Tucker.]

Each processor in CM contains:

1. an Arithmetic and Logic Unit (ALU), and associated latches,
2. four (sixty four for CM2) Kbits of bit-addressable memory,
3. eight one-bit flag registers,
4. a router interface, and
5. a two-dimensional-grid interface.

Operations are executed in a bit serial fashion. An ALU consists of a three-input two-output logic element and associated circuitry. During an execution cycle, two bits are read from memory and one bit from the flag registers. Then, the ALU computes two result bits, one bit is stored back into memory and the other is used to update the flags. Despite this simple logic structure CM is able to carry out all the operations of a virtual-machine instruction set - complex instructions are decoded by the sequencers which control the ALUs.

CM contains a flexible inter-processor communication network which supports several mechanisms:

- a broadcast facility allows immediate data to be sent to all the processors from either the front-end computer or the sequencer.
- Global logical OR allows the outputs from all processors to be checked simultaneously for termination conditions.
- Hypercube communication forms the basis for the router and other primitives. The topology is a binary 12-cube connecting 4096 nodes[5].
- The router implements general pointer following through a packet switching scheme. The router controller (hard-wired into the CM processor chips) uses the 12-cube for data transmission.
- The North East West South (NEWS) is a two-dimensional mesh which provides direct communication between a processor and its four neighbours.

Figure 3.2 shows the CM system organisation. Each CM1 sequencer is a purpose built micro-computer used to implement the virtual machine. It is built around Advanced Micro Devices bit-sliced micro-processors with 16K 96-bit words of micro-code storage.

---

[5]Each node consists of 16 ALUs and support hardware in a single proprietary chip.

It is believed that the high performance[6] achieved by CM can be attributed mainly to the extensive support it affords to the various communication requirements of algorithms. Besides, the virtual machine implementation takes the user away from the intricacies of synchronisation and scheduling. Finally, the high bandwidth of the communication paths between the front-end computers and the sequencers ensures that when computation is needed and processors are available, minimal delay is incurred.

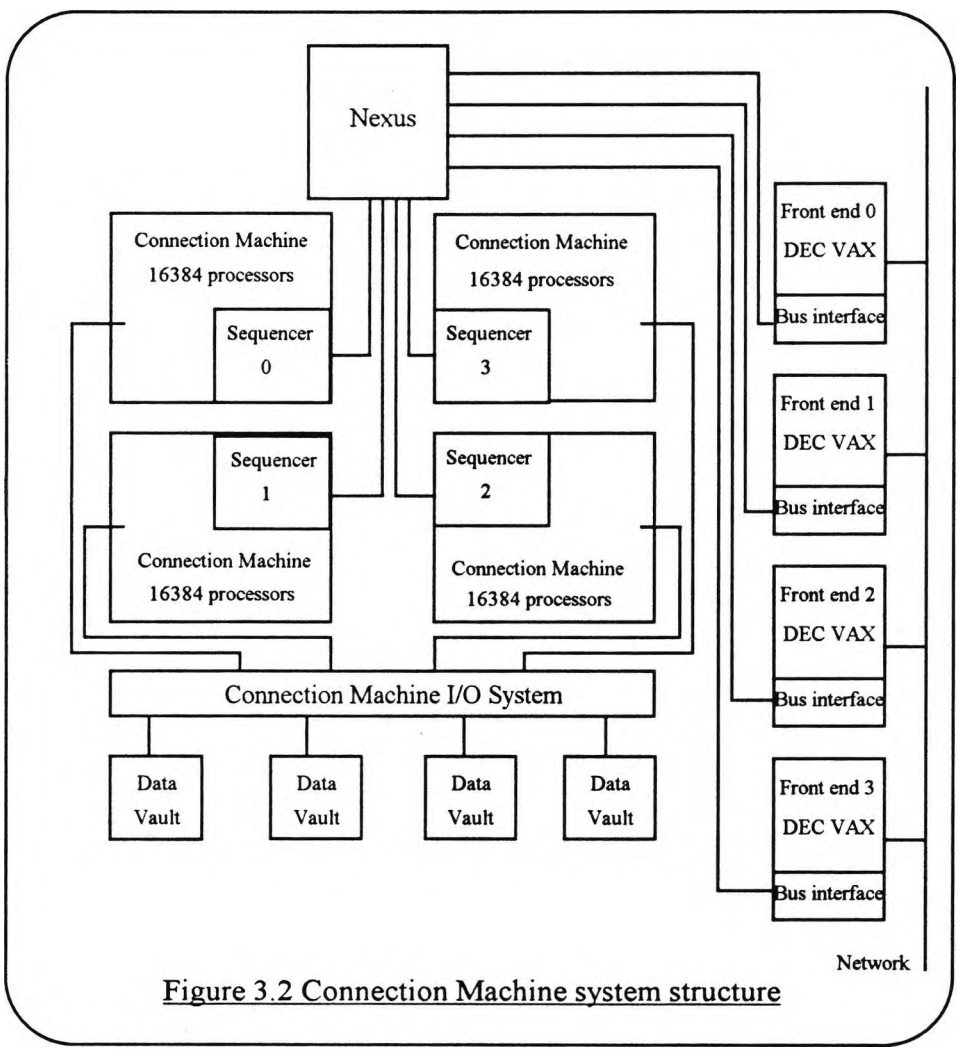

Figure 3.2 Connection Machine system structure

The CM has been used in a number of applications ranging from the regularly structured problems of materials science to artificial intelligence and computer graphics [Tucker][Waltz]. In one instance [Tucker 2] an object recognition system was

[6]With 64k processors working in parallel CM1 achieves an aggregate rate of 2 billion 32-bit integer additions per second.

developed on the CM. The approach adopted by Tucker *et al* was the massively parallel hypothesis generation method. In order to avoid constraint-based tree searches which would have been too taxing in terms of communication overheads, hypotheses are generated when features belonging to objects in a database match features in the image being analysed. One of the most important aspects of the CM as far as algorithms with time varying communication requirements was reported to be the support for both nearest neighbour (NEWS) and general communication schemes (Hypercube).

It is worth noting at this point that Thinking Machines Corporation's (TMC) latest product (CM5) is a message-passing MIMD system. Each node of CM5 comprises a SPARC microprocessor and optionally four vector units between the processor and its local memory [Gottlieb]. Therefore, CM5 does not fit into the army-of-ants class; and TMC's approach in trying to achieve higher performance is akin to that of the next section.

3.4.2. The herd of elephants approach

In contrast with the army-of-ants approach where processors are only capable of performing very simple tasks, this approach requires individual processors to be 'powerful' computers in their own right. Clearly, the adjective 'powerful' is very subjective; in the remainder of this section it will be deemed to describe current state-of-the-art micro-processors. Another difficulty consists of the fact that the Tera-Flop (10^{12} floating point operations per second) machine does not seem too far in the future [Langhammer][Astfalk]. Such a computer will have to be implemented using a massive number of processors making the 'herd' as large as an 'army'. However, to achieve the same performance, a CM-type computer will require two to three degrees of magnitude more nodes; thus, keeping the distinction between the two approaches.

In the following sections, two commercially available parallel computers are presented. First, Convex computer Corporation's MPP range is described. Then, Parsytec Computer Gmbh's GC range is introduced. The former is based on a Hewlett-Packard PA-RISC micro-processor and the latter uses the latest Inmos transputer (T9000).

3.4.2.1. Convex MPP

The Convex range of Massively Parallel Processors is designed to be scalable to TeraFlop performance. MPP has a globally shared virtual memory MIMD architecture.

Each node consists of a Hewlett-Packard PA-RISC microprocessor running at 100 MHz and providing 200 MFlops of peak performance.

One important aspect of MPP is the emphasis on ease of use. Convex has been developing a compiler that performs inter-procedural Optimisation [Astfalk]. The basic idea revolves around extending the optimisation phase (i.e. optimisation confined to loops and loop nests) with techniques that identify larger granularity parallelism.

3.4.2.2. Parsytec GC range

The current computer in the GC range is GC5 [Langhammer]. The processing nodes are the latest transputers (T9000 - appendix A). The proposed GC5 will contain 16384 transputers connected as a message-passing MIMD parallel system. It is organised in a three dimensional grid of atomic cells. Figure 3.3 shows an atomic cell. The latter consists of 16 transputers connected as a 4-dimensional hypercube, a redundant transputer to replace any other on failure and 4 routing chips (Inmos C104).

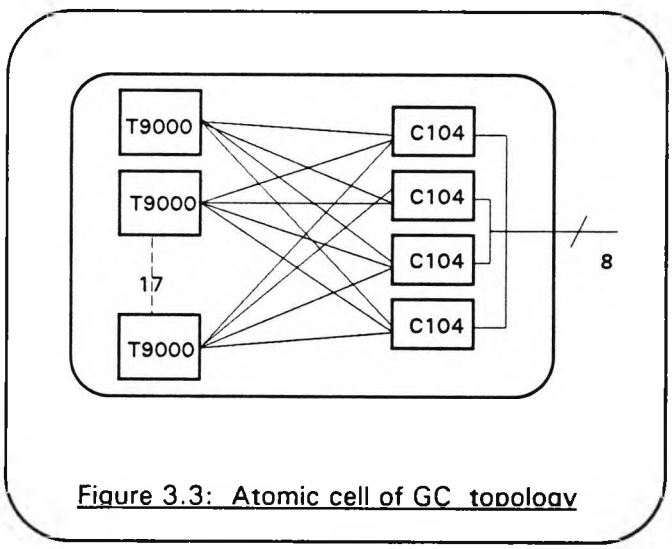

Figure 3.3: Atomic cell of GC topology

Amongst the advantages of using the transputer as a building block for massively parallel processors are:

- Availability of all components (i.e. there is no need for custom design).
- Because the transputer incorporates a floating point unit, a cache, memory glue logic and communication and routing support, node design is simplified.
- A micro-coded scheduler allows the processor to be shared by several processes at two levels of priority.

3.5. Discussion

To summarise, manufacturers and researchers agree that a crucial factor in the effective use of parallel computers and in particular massively parallel processors is the interconnection scheme adopted. A balance has to be established between the memory bandwidth and the communication bandwidth. Hence, the (important) choice of the unit node defines the model of parallelism to be adopted. The MIMD model has in recent years been presented as the means to achieve high 'general purpose' performance (CM5, Convex MPP, Parsytec GC, Cray MPP, etc.)
Many algorithms that are inherently parallel have a local communication behaviour. Therefore, they are well suited for 2D and *n*D array structures. However, algorithms in general and Computer Vision algorithms in particular have varying communication requirements and granularity. In chapter 7 a few well known topologies are analysed and a configuration suitable for the transputer (T800) is proposed as an attempt to provide both the regular structure that benefits data-parallelism and relatively small distances (for a given number of nodes). The latter, together with a dynamic reconfiguration scheme, benefit data broadcasting and functional-parallelism. A routing algorithm is presented in chapter 8.

Before finishing this chapter standards ought to be mentioned. With the large number of research and commercial systems being proposed, standards for parallel computing have become necessary. One such standard, is the Message Passing Interface (MPI) [Gropp], is of particular interest to this work, because it is specifically designed for applications running on distributed memory concurrent computers.

CHAPTER IV

4. Transputer implementations of edge detection

This Chapter presents some work on low-level image processing using transputers. It consists of slightly modified versions of two publications [Omar 2][Ellis], and is organised as follows. First, an adaptive approach to edge detection is given [Omar 2]. Then, a method that uses colour information to improve the edge detection process is presented [Ellis]. The work was carried out in collaboration with colleagues at the Centre for Information Engineering at City University.

4.1. Transputer implementation of adaptive noise cancelling

4.1.1. Introduction

The work presented in this section uses an adaptive approach to noise cancelling in digital images. This improves the performance of a subsequent edge operator. Noise filtering is necessary as a pre-processing stage for edge detection [Canny]. By using an adaptive filter, assumptions usually made about the nature of the noise in the image are relaxed. This leads to an improved performance over a wide range of input images.

The algorithm implemented consists of two linear noise cancelling filters cascaded with the transpose of their derivatives. The output thus produced, is passed through a non-maximum suppression process which ensures a single response. The resulting image is thresholded using a smooth non-linear function.

Inherent parallelisms are identified and exploited through the use of a transputer network. Data parallelism leads to partitioning of the image into small segments that are processed independently. Further parallelism can be identified in the algorithm, since the horizontal and vertical kernel can be applied to the image separately.

4.1.2. Signal processing and digital images

One of the first stages in image processing is the extraction of information from an image, presented in digital form as a matrix of intensities in quantized (grey level) form. At this level the image may be regarded as a two-dimensional signal containing information such as objects on a background. These are often corrupted by other signals, such as noise.

Noise affects the reliable extraction of features, like edges from an image. Most edge detectors use gradient information to detect boundaries of objects and are particularly susceptible to "high-frequency" (spiky) interference. It is therefore necessary to reduce or, if possible, remove the noise before edge detection can take place.

4.1.2.1. Noise cancelling

Noise is often removed by applying a filter to the image. The filter coefficients are selected to remove as much of the noise as possible, by smoothing the image whilst maintaining the desired image features unchanged. Failure to select the appropriate coefficient values may result in either high levels of noise or excessive smoothing. Both effects can be hazardous for subsequent edge detection since they may lead to either false detections or missed edges.

If the characteristics of the noise are known beforehand, the filter can be tailored as described. This is often not the case. It is, however, possible to start from an initial guess for the coefficients and correct them when the image, and hence the noise characteristics, become available.

To determine the optimal noise cancelling filter, the study of the signal through a model is often useful.

4.1.2.2. A model for the image

Although an image is a two-dimensional signal, its horizontal and vertical components can be studied separately. For edge detection purposes the cross-section of an edge is almost constant in any direction, for small steps in the perpendicular direction.

Edges usually signify boundaries between regions of interest, such as objects. In a grey-level image, these are distinguished from their different intensities. Objects normally appear as transitions in the intensity of the image that last for several pixels. Noise, on the other hand, consists of random transitions in the intensity that flood the image and have no structure. The noise-free image signal S(i,j) and noise N(i,j) are added to form the recorded image I(i,j) (figure 4.1(a)):

$$I(i,j) = S(i,j) + N(i,j)$$

N(i,j) is normally treated as a stochastic process with fixed statistical properties throughout an image. The best noise cancelling filter should reflect these properties.

4.1.2.3. The adaptive solution to optimum filtering

If a signal d(n) is to be produced from another signal $x(n)$ by linear weighting,

$$y(n) = \sum_{i=-N}^{N} w_i x(n-i) = \vec{W}\vec{X}(n)$$

a cost function, such as the mean squared error

$$J = E\left[\left[d(n) - y(n)\right]^2\right]$$

must be minimised. The best weight (coefficients) values $\vec{W} = (w_{-N}, \ldots, w_0, \ldots w_N)^t$ satisfy the Wiener-Hopf Equation

$$R_{xx}\vec{W} = \vec{R}_{dx}$$

Where R_{xx} is the auto correlation matrix of x(n) and R_{dx} is the cross correlation vector between x(n) and d(n).

In a noise cancelling situation, x(n) is the linear combination of the wanted signal s(n) with uncorrelated additive noise n(n):

$$x(n) = s(n) + n(n)$$

The objective is to find W such that y(n) is the best approximation of s(n). If d(n) is not available separately, like in our case, x(n) can be used as a noisy estimate of it.

The Wiener-Hopf Equation can be solved recursively, e.g. using gradient search techniques[Widrow][Orfanidis]. A simple recursion equation for W was described by Widrow and Hoff [Widrow 1]. It is known as the Least Mean Squares (LMS) algorithm:

$$\bar{W}(n+1) = \bar{W}(n) + 2\mu e(n)\bar{X}(n)$$

where

$$e(n) = x(n) - y(n)$$

$$y(n) = \bar{W}(n)\bar{X}(n)$$

$$\mu = \lambda / tr(R_{xx}) \quad (0 < \lambda < 1)$$

μ is the adaptation step and governs the rate of convergence of the filter. Small μ causes slower adaptation but smaller error in the final $\bar{W}$.

4.1.3. Adaptive noise cancelling and edge detection

Most of the adaptive algorithms were initially developed for time sequences. The LMS algorithm is not linked with time-related data and is therefore suitable for applying to image data. There is however a fundamental difficulty in the adaptation procedure. Whereas in a time sequence there is a natural and compulsory direction for filter application and adaptation, (i.e. forward time), such a direction does not exist for a single image. A good procedure is to select randomly the next location in the image on which the filter is to be applied. This method, however, suffers from two practical limitations. First, the adaptation and filtering stages have to be separated, to guarantee that all image elements are processed. Second, localisation of filter coefficients could be lost. Last, noise characteristics may differ from one part of the image to another. Better results are to be expected from locally optimal filters than a globally optimised filter over a large area. For this reason the filter is applied in the horizontal and the vertical directions.

Initially, a two-dimensional (NXN) filter was used for smoothing. This required $2N^2$ operations and did smoothing in both directions simultaneously and resulted in rounded corners. Subsequent edge detection had to be completely separated from the filtering operation.

A different approach proved more effective both in terms of number of computations necessary and in terms of edge detection. It was based on a commonly used filter structure in image processing, the use of two one-dimensional filters to reduce noise in both the horizontal and the vertical directions. Typically, these filters have fixed coefficients based on assumptions about the nature of the noise. Gaussian filters are among the commonest. The variance of the Gaussian function, which determines the amount of smoothing, is specified at the beginning of the filter process and is not modified afterwards. Each filter produces its own output, which is used for further processing.

The proposed method of smoothing is similar, but the fixed-coefficient filters were replaced by adaptive ones. The coefficients of the two filters were initialised to either zeros or to those of an initial estimate of a Gaussian filter. Both methods performed almost identically. An example of a smoothed image profile is shown in figure 4.1(b).

4.1.3.1. Edge detection

Usually edge detectors are fixed-coefficient differential operators. The differentiator presented here uses the derivatives of the adaptive filters after adaptation is completed. Each of the differentiated filters was then applied to the smoothed image produced by the other filter, like in the Canny operator. The output of this operation is a set of two images, with sharp transitions emphasised and the rest of the image suppressed. Because the other source of large gradient values (i.e. noise) was already suppressed, the remaining large gradients are expected to indicate edges.

4.1.3.2. Gradient detection

The two differential images produced in the previous section were then combined to detect edges in any direction. The horizontal and vertical differentials at every point may be regarded as a vector of two orthogonal components. The modulus of the vector was taken as a measure of "edgeness". The "direction" of the edge was also

determined as angular deviation from the horizontal, although this feature was not used.

4.1.3.3. Non-maximum suppression

The differentiated image was then treated further to produce a binary image indicating whether a point was classified as an edge or not. A good edge detector should not only detect the existing edges, but perform good localisation too. Part of the second process is a single response to an edge.

A simple non-maximum suppression algorithm produced good results in terms of single response. A candidate edge is not suppressed if it is a local maximum in any direction. This test involves evaluating its gradient modulus against that of its eight neighbouring locations. A further test is then performed to determine if the gradient is high for the area where the point lies, by passing its gradient, g(i,j) through a sigmoidal non linearity where the threshold ϑ is the mean and its exponentiating factor σ is the standard deviation of the gradient in the neighbourhood:

$$f(g(i,j)) = \frac{1.0}{1+e^{-\sigma(g(i,j)-\vartheta)}}$$

Only if the obtained value is over 1/2, the pixel at (i,j) is considered to be an edge (figure 4.1(c)).

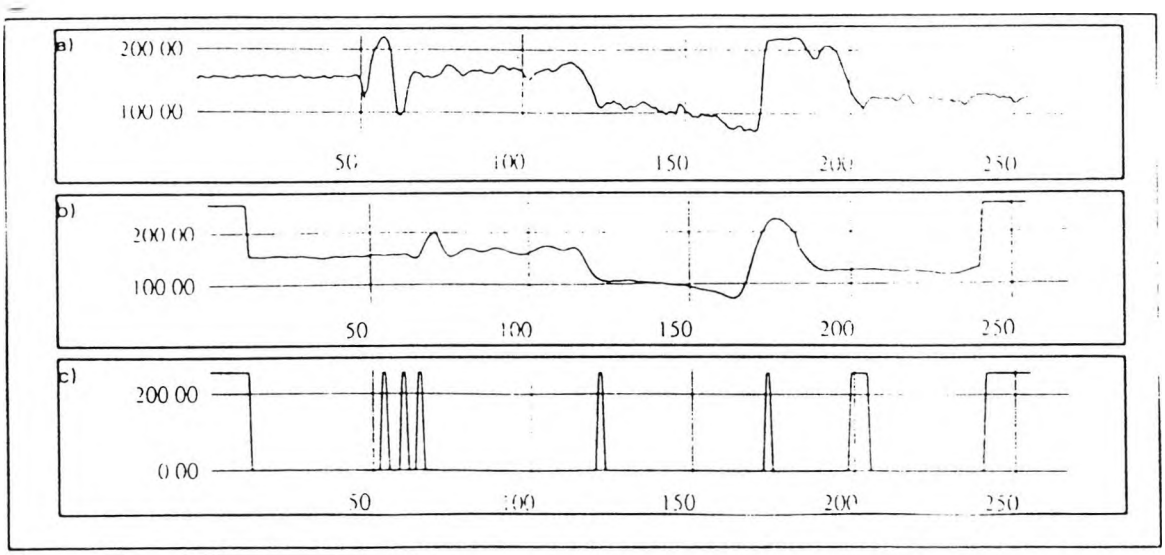

Figure 4.1 One scan line from the image

4.1.4. Implementation

In this section the algorithm is described and a transputer implementation is proposed. First, the potential parallelisms are identified; then a mapping of the algorithm on a network of processing elements is presented. The analysis is based on a message passing framework, suitable for transputer modelling.

4.1.4.1. Algorithm description

The ideas behind the algorithm can be derived from the definition of the procedure adopted. First, because operations in different areas of the image are independent, data can be shared amongst a population of processors. Next, the unit process is identified in order to take advantage of any inherent functional parallelism. The notion of identifying independence derives from the necessity to limit the amount of communication among processors, and try to achieve induction i.e. linear speedup.

4.1.4.2. Inherent parallelisms

As mentioned above the system consists of an adaptive noise canceller, cascaded with a local gradient operator, obtained by differentiating the filter coefficients. The output of the latter is fed into a non-maximum suppression unit that enforces single response. Figure 4.2 shows a graphical representation of the processes that implement the system.

4.1.4.2.1. Data parallelism

Adaptive noise cancelling relies on the ability of a variable kernel evaluated from a neighbourhood of pixels around the sample to estimate the signal. This completely local dependency suggests the partition of the image into independent blocks that do not require external data. Because the noise canceller and the subsequent gradient operator are convolution operations, half a mask size overlap has to be packed with the data for each block. Although this presents an overhead, it is a major set back only for very small blocks. At this point, the important issue in mapping such a partition on a network of processors is to minimise the distances travelled by data packets, bearing in mind that the real distance is not only the number of hops that data packets makes but the delay incurred during routing as well.

4.1.4.2.2. Functional parallelism

The different stages of the algorithm (figure 4.2) are:

- Horizontal smoothing,
- Vertical smoothing,
- Transposition,
- Horizontal gradient,
- Vertical gradient,
- Non-maximum suppression and thresholding.

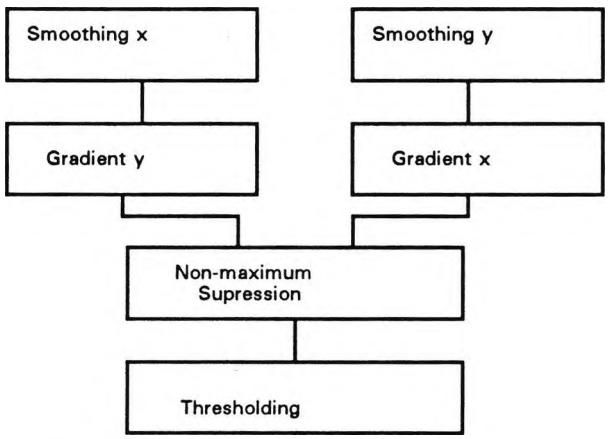

Figure 4.2 Block diagram of the system

The horizontal and vertical smoothing operations are independent and can be carried out in parallel. The transposition operation refers to the fact that the gradient operator in one direction is applied to the output of the smoothing operator in the other. The outputs of the gradient operations are combined to produce the combined gradient map. This leaves the non-maximum suppression and thresholding as a final stage. This can be implemented as three separate stages.

4.1.4.3. Transputer Configurations

The networks described here shows two types of parallelisms mentioned above. First a simple linear structure is given, then the functional parallelism is incorporated to yield a more efficient architecture.

4.1.4.3.1. Simple linear structure

A simple linear structure of sixteen transputers was implemented to take advantage of data parallelism. More complex networks have not been considered because of the large processing to communication ratio. The further performance improvement that can be achieved for such a small number of transputers will be marginal. Figure 4.3a shows this simple structure, while figure 4.3b presents the processes inside each transputer in the network. The root transputer interfaces with the host system (a Sun Sparc station) and provides data partition services and overlap calculations for the rest of the network.

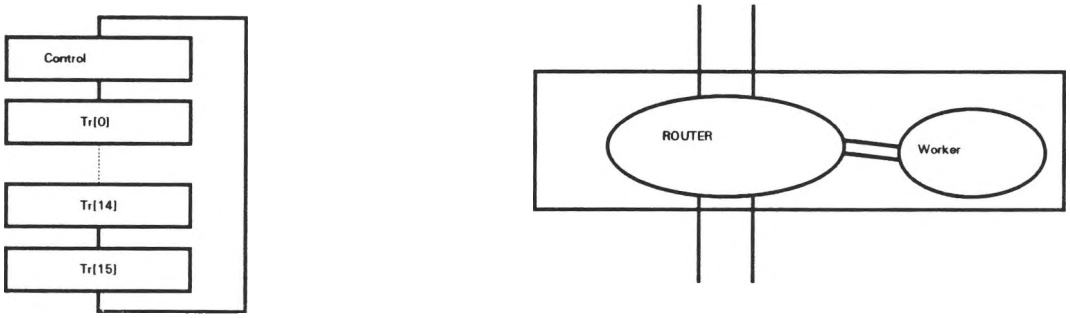

Figure 4.3 a) The simple linear structure, b) Inside Tr[i] i = 0 .. 15

4.1.4.3.2. Multi-level pipelines

The algorithm as described above decomposes into three separate stages. However, an efficient implementation cannot consider the third stage (i.e. non-maximum suppression and thresholding) as an element of the pipeline since this process is far less complex than the other two stages.

The structure proposed here consists of series of simple two stage pipelines combined with a third processor to form one level in the structure. Levels are stacked to form the overall network. The third processor in each level is used to run the simpler final stage, and route the results back to the host system (figure 4.4).

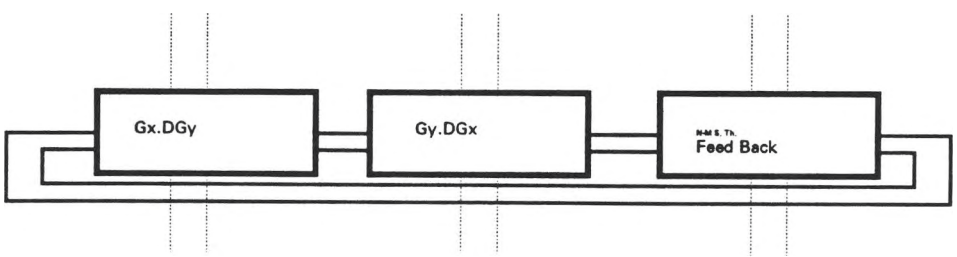

Figure 4.4 Unit structure for multi-level pipeline

4.1.5. Results

Results of the edge detector are shown for two sample images in figure 4.5 (c,d), together with the original images (figure 4.5 a,b). The output shows single response to different types of edges and a good localisation.

Filter coefficients after adaptation (figure 4.6) seem to confirm the validity of using a Gaussian model for additive noise in digital images. However, this noise is not stationary. The adaptive filter relaxes stationarity and 'zero-mean' conditions. Figure 4.7 shows the modulus of the weights vector during and after adaptation.

A speed-up of 12.5 was registered with a sixteen transputer network (see section 4.1.4). Further improvements are anticipated since the main processing load which consists of the two convolution processes is partitioned. The additional communication due to passing the same block to the two processors in the pipeline stage is countered by the fact that only coefficients are passed between processors, after smoothing.

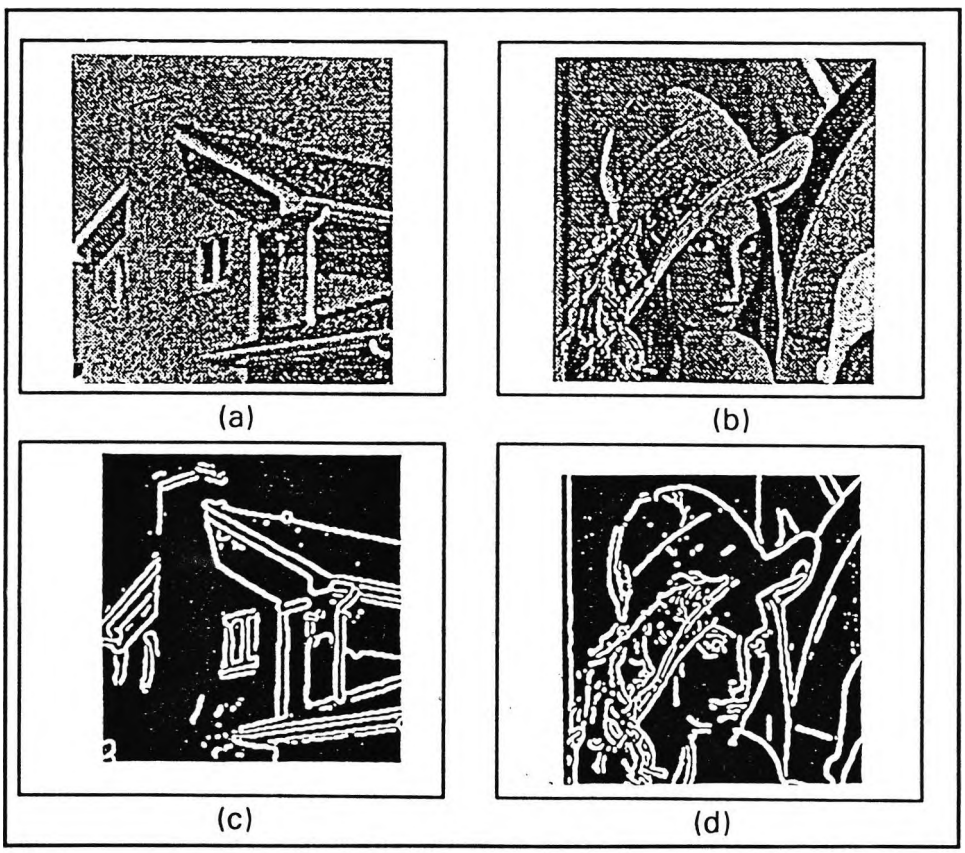

Figure 4.5 Original and processed images

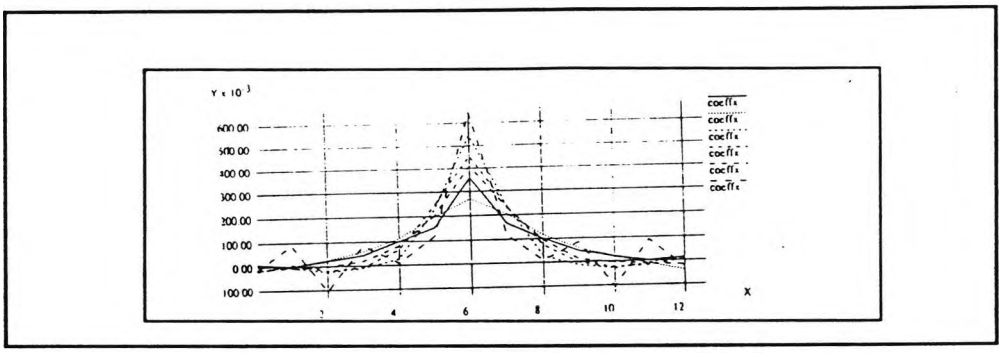

Figure 4.6 Filter profile after adaptation

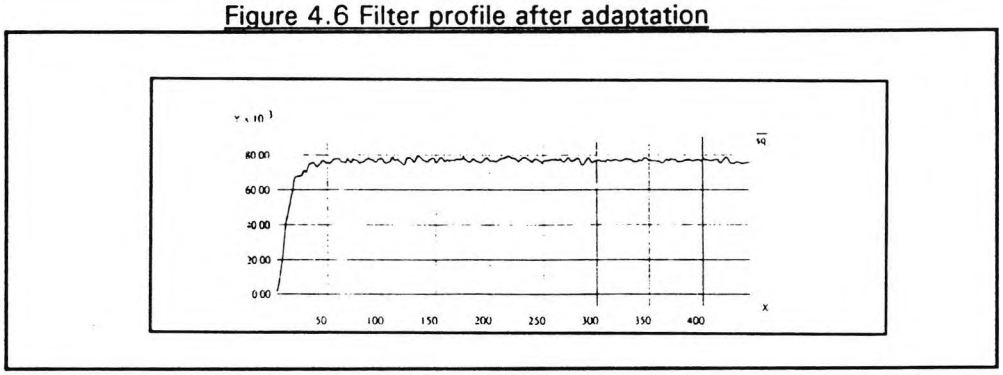

Figure 4.7 Modulus of weights vector during adaptation

4.1.6. Comments and conclusions

The system presented here implements an adaptive noise canceller and edge detector. The results show a good localisation of edge elements and immunity to noise. The transputer implementation showed speed ups which suggest that further improvements are possible if more processors are added to the system.

An interesting observation was made when analysing the filter coefficients. The kernels seem to latch on to textured areas of the image. This, together with the good immunity to noise, suggests that adaptive filters can be used as a learning process for texture registration. The filter coefficients thus obtained can form cluster centres of a parameter space to search.

4.2. Colour edge detection

This section describes the transputer implementation of a system for the extraction of edges from colour images. The early processing stage of the algorithm is based on the conjecture which states that real edge pixels will generate similar gradient angles on the three planes of a red, green and blue (RGB) image. The first stage of the edge detection process uses an adaptive method for selecting the level of smoothing applied to the image. The computational cost imposed in analysing RGB images is met by a

parallel implementation of the algorithm, and issues raised in data and process partitioning are considered.

4.2.1. Introduction

The majority of machine vision systems make little or no use of colour information in analysing scenes. This is attributable to a number of reasons:

- colour constancy - perceived colour depends on the spectral content of the illuminating radiation, as well as on the spectral reflectance characteristics of the surface, and hence does not provide a robust feature for object detection.

- high-resolution colour cameras are expensive and bulky.

- processing a colour image (e.g. 3 images for RGB) imposes additional computational burden, and is less likely to be used if only a small benefit can be attained from the use of colour.

- edges in colour images have not been found to be significantly 'better' than for monochrome images [Nevatia].

Nevertheless, colour is used in a number of systems; in particular, where colour information must be explicitly represented (e.g. colour coded resistors [Claxton]).

This section describes a method for using correlation of edges over the RGB planes of a colour image to supplement the edge detection process commonly used in monochrome images, and the use of an adaptive smoothing filter to suppress image noise, prior to edge detection. These techniques are used as a pre-processing operation to a system for the labelling of structural elements (lines, curves, corners) in the colour images. The correlation of edges can offer additional confidence in the existence of an edge element in one plane when it is corroborated by one or more pixels from the other colour planes.

The algorithm consists of four parts. First, a pre-processing stage estimates the local information content of the image (local entropy). This is then used to adjust the standard deviation of a modified Canny edge detector [Canny] which constitutes the second stage.

This edge detector is run in parallel on the three planes of a colour image, and evidence of edges detected in one plane must be corroborated in either (or both) of the other planes to indicate a reliable edge point. The next stage links the edge elements into connected structures, based on a simple 8-neighbour connectivity. The fourth and final stage identifies straight lines from curves and locates corners in the connected edge data by accumulating consistent gradient direction information. Following a brief description of the algorithm, the transputer implementation of its processes is described.

4.2.2. Pre-processing

The aim of this step is to provide an estimate of the local information content in the image. The method is summarised as follows: In each 4x4 pixel block evaluate the gradient in horizontal, vertical and diagonal direction normalised by the average intensity.

The block is classified as an edge block if the gradient exceeds some threshold in one of the directions. This gives a crude estimate of the edge location across the image and is used to guide the selection of the standard deviation used in the edge detection algorithm. The standard deviation (sigma) of the Gaussian smoothing is varied according to the number of detected edge blocks in a given region (currently rectangular) and ranges from 1.0 to 3.0 in an inverse proportion to this number.

4.2.3. Colour Edge Detection

Researchers have in the past investigated the use of colour in using simple edge detectors (e.g. Roberts, Sobel) applied to the different image planes of multi-spectral images to improve detection [Robinson 1]. The major problem is how to combine the outputs in order to interpret the edge information. Nevatia [Nevatia] found that accumulation of the gradient output of the Roberts operator does not lead to better results. It is believed that inconclusive results from previous experiments are due to poor edge operators and to output combining functions. The method presented here is based on the use of an improved edge operator (as described above), together with the following conjectures:

- A real edge is likely to produce high gradient magnitudes on the three image planes.

- The gradient direction should be within some tolerance (modulo π) on the three image planes. Therefore, the presence of an edge element at a particular location in one plane can be corroborated by a consistent registration at the same location on another plane.

4.2.4. Line and Corner Extraction

In this part edge elements are linked into connected structures. The usual step of isolated pixel elimination is not necessary because of the strong emphasis on edge detector. Edges are followed and accumulated. Junctions are resolved by following the closet gradient angle and starting new structures for the other branches. Lines are separated from curves by considering proximity and gradient angle. After separation a corner finder is applied to the set of lines found. This consists of an intersection (or proximity of end points) test. The overall result is that an "accurate" edge map constituting the basis for structural element detection. This process is interleaved with the rest of system. In other words, once an edge element is definitely registered, it is linked to a previously created linked lists of edge pixels if it is connected to them, or a new instance of the data structure is created for it. At this stage, the classification of the structural element as a line, an arc or a corner is not considered. However, junctions and splits are resolved by accumulating the connected set that has the smoother gradient angle changes. The next step is the classification of structural elements. To do this, the curvature of the connected set within some tolerance discriminates between linear structures and curves. Corners are considered to be points of discontinuity in the gradient angle. A connected structure is sectioned at these corners into three parts: two linear structures and a corner. It is believed that classified elements like the ones described above are useful as an input to higher level processing. (Refer to [Ellis] for implementation and results).

4.2.5. Conclusions

A scheme for combining information in multi-spectral images is presented and used to optimise the output of an edge detector. The use of colour does give some clear improvements in the edge detection process. We are currently combining this with adaptive smoothing in order to investigate the interaction between the two operations.

The edges thus detected are linked into structures which in turn are partitioned and classified. The implementation of the system can be improved in many ways:

- through the implementation of structural element classification across the network.
- through a more formal view of the process as a graph and the use of graph theoretical results together with the transputer idiosyncrasies.

CHAPTER V

5. Parallel vector quantisation

5.1. Introduction

This chapter presents a parallel algorithm for Vector quantisation (VQ). VQ has been shown to be very effective in the compression of both speech and image data.

The implementation of image coding based on VQ can be divided into three stages:

1) Design of the codebook (s)
2) Encoding of the image
3) Decoding

Stage 1 is a lengthy computational process, but is only performed infrequently. Stage 3 is a simple lookup process: the code generated (typically a single integer) is used to directly address the stored vector associated with it. However, stage 2 requires a search through all the code vectors and matching to the best vector by calculating a distortion measure between the candidate image block and the code vector. Hence the encoding process dominates the speed at which images can be encoded and decoded.

Vectorisation which is the first step of the encoding process refers to the partitioning of the image into contiguous, non-overlapping blocks. Square blocks are generally preferred because they are more convenient to deal with and easily generated.

Quantiser (codebook) design refers to the generation of the reconstruction vectors Y_i. The set or collection of reconstruction vectors is called a codebook. An optimal vector quantiser is one which employs a codebook C that yields the least average distortion D* for all such codebooks. The design algorithm for such an optimal codebook is not known. Quantiser design is a very difficult problem in vector quantisation. It is generally a non-linear problem involving a lot of iterative computations. Clustering algorithms such as the LBG algorithm of Linde, Buzo and Gray [Ramamurthi] are used to obtain locally optimal codebook designs. The existence of an optimal codebook is assumed; therefore, we concentrate on the encoding process.

The encoder and decoder have identical codebooks. The input vector X to the vector quantiser is a vector of dimension k. The encoder computes the distortion $d(X,Y_i)$ between the input vector X and each codevector Y_i, i=1,N from a codebook C containing N code vectors. The optimum encoding rule is the nearest neighbour rule [Ramamurthi] in which the index I is transmitted to the decoder if codevector Y_i yields the least distortion.

Decoding involves the decoder looking up the I-th reconstruction codevector Y_i from its copy of the codebook to reconstruct a replica $X = Y_i$.

5.2. Computational complexity of VQ encoding

The distortion measure considered is the mean squared error (MSE) :

$$d(X,Y_i) = (X - Y_i)^t(X-Y_i) \qquad (1)$$

where X is the input vector, Y_i a reconstruction codevector and the superscript t denotes transposition.

Consider codebook C with properties as in section 5.1. For full search optimal encoding of vector X, the distortion must be computed for each of N codevectors in the codebook. The number of arithmetic operations involved per codevector is 3k - 1 (k multiplications, k subtractions and k-1 additions). Thus, for all N codevectors the number of arithmetic operations is (3k-1)N. For an M by M image the number of operations is given by

$$O = (3k-1)N.M.M/k \qquad (2)$$

The bit rate of the coder (in bits per pixel) is given by

$$R = (1/k)\log2(N) \qquad (3)$$

Thus $N = 2^{kR}$. Substituting for N in (2) gives

$$O = (3k-1)*2^{kR} * M*M/k \qquad (4)$$

It is thus seen from (4) that the number of operations grows exponentially with both k and R. The computational complexity of the encoding process (as well as the codebook design) tends to restrict vector quantisation to small block sizes (e.g 4 by 4) and small bit rates and to still or slowly varying frames. In this simple approach although a small bit rate is interesting from the compression point of view it limits the size of the codebook for a given block size and thus affects the quality of the reconstructed image. Various methods have been developed to eliminate or reduce the exponential growth of the computational complexity. Reductions in computational complexity are achieved by modifying the codebook, by sacrificing performance in achieved average distortion, and/or by increasing the storage requirements of the codebook (e.g. in the binary search tree codebook). As far as the quality of the output is concerned a great deal of effort has been put into trying to achieve better perceptual quality (e.g. Classified Vector Quantisation (CVQ)).

The aim here is not to reduce the computations nor to increase optimality regarding coding distortion but to attempt to increase the speed of the full search encoding process by means of parallelism using transputer networks. Some topologies are proposed for VQ and CVQ.

The transputer topologies considered are described in the following sections.

5.3. The use of parallel processing (transputer)

Among the major concerns of parallel computers are the speed of computation of the separate nodes and the speed of communication (between nodes). At the lowest level, given that a choice of hardware is made at the outset (namely the transputer), these two variables seem to have been fixed. However, the techniques used in implementing the problem, together with the efficient use of the capabilities of the transputer can have considerable consequences on the system.

The key feature of the vector quantisers presented here is that the vectorisation of the image yields non-overlapping blocks. Thus the mean square distance calculation at the centre of the coding process is easily identifiable as the unit building block of the quantiser. There seems to be no point in further sub-dividing this unit block since the time required to perform the operation will approach the time needed to gather the sub-results due to link set up and routing. N (the size of the codebook) of the above mentioned building blocks are needed to code one vector of the input image. Hence there is scope for sharing this task among a few transputers. However, these two potential parallelisms are interdependent in the case of VQ. Consequently CVQ, which achieves better quality by classifying the codebook (hence partitioning) was considered.

Networks consisting of a ring and a two dimensional mesh were implemented to take advantage of the image data partitioning mentioned. Higher dimensional structures were not considered because of the cardinality of the data dispensing node (frame grabber). The critical notion of distance travelled by data packets can only be reduced in real terms by introducing more elaborate input and output mechanisms.

Several experiments were conducted using a farm topology and a distributed monitor (based on [Jones 1]) to measure the busyness of individual transputers in the network. Up to 10 can be used in bilinear farms without sizeable impairment to the rate of speedup improvement. Higher order farms were not considered because of the need to integrate image data partitioning in the implementation of CVQ and the fact that their benefit shows only with larger number of transputers.

For a given codebook and image size and with the distance measure operation considered atomic, the improvement anticipated is given by :

$$\frac{1}{\frac{1}{n*m} + d_{comm}} \qquad (4)$$

where n is the size of the codebook partition and m is the size of the image data partition. d_{Comm} is the equivalent number of operations for the time taken by the overall communication overhead. In CVQ however the codebook partitions size is not constant. Hence, a different sub-network for each class should yield better results (refer to [Omar 1] for details of implementation and results).

5.4. Conclusions

A significant improvement in the time taken for the coding process of VQ and CVQ was achieved through the use of relatively simple networks of transputers. Limiting factors to the scalability of the structures presented here seem to be the communication bandwidth and the real term distance from the data dispensing node and the working nodes.

In the case of CVQ, statistics show that some classes are more populated than others. To ameliorate the performance of the network introduced here, different sub-networks should be designed to cater for the unbalanced load.

It is worth noting that these structures can be further investigated in order to find the practical limit to scalability, and then implemented on a memoryless transputer network (+frame grabber) as a fast cost effective coder. The size of partitions will, in this case, be dictated by the transputer's on board memory.

CHAPTER VI

6. Transputer implementation of some vision tasks

This chapter presents the transputer implementation of three tasks common in computer vision. These tasks were chosen because of their differing communication requirements and amount of inherent parallelism. First, the Fourier descriptors method is presented and a parallel algorithm proposed. Second, the Hough transform technique is introduced in both original and generalised form. Last, matching objects in a scene to computer models is considered and the 'maximal clique' technique introduced. Several attempts at reducing the sequential part of this algorithm are presented.

6.1. Fourier Descriptors

Classification of objects in a scene is one of the problems of image processing. Fourier descriptors represent the boundary of a region as a periodic function which can be expanded in a Fourier series [Persoon]. Several different parameterisations will be presented, these are due to several workers [Persoon][Waltz][Duda]. The aim of this section is to study the relative merits of various boundary representations and to present an algorithm well suited for parallel implementation.

Fourier descriptors provide an improved characterisation of shape as more coefficients are added to the frequency domain sequence. In the limit of an infinite sequence they are completely unambiguous. Individual coefficients describe the boundary with increasing accuracy.

Fourier descriptors do not offer easy reconstruction of the space domain shape. This is due, for example, to the fact that a finite frequency domain representation will inverse transform into an incomplete spatial description of the periodic function that gave rise to it.

One of the major advantages of Fourier descriptors is that the general shape of a boundary can be described satisfactorily by a few of the low-order terms in the Fourier series expansion of the boundary curve. Besides, a well chosen parameterisation can lead to a frequency-domain representation that is independent of size, translation and rotation of the shape to be described. This can have very interesting consequences on the storage of parameterised shapes, yielding a reduction in the communication overhead they introduce in a distributed implementation of shape recognition.

6.1.1. Parameterisations

In this section two parameterisations are considered. First, a closed curve is represented by the cumulative angle between a fixed line in the image and the curve figure 6.1 (a)). Second, the pixels around the boundary are considered as points in the complex plane (figure 6.1 (b)); therefore, scanning the boundary generates an array of complex numbers. Both methods rely on the perimeter of the curve to fix the sampling rate. It is worth noting that the first parameterisation yields a sequence of real number making it more attractive from the computational point of view. The next few sections will present both methods and a comparison of their merits and drawbacks.

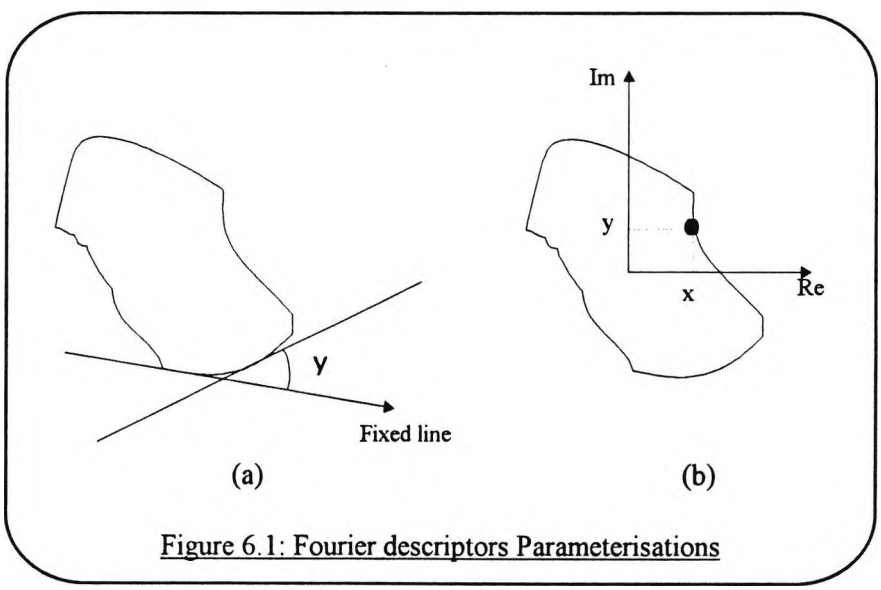

Figure 6.1: Fourier descriptors Parameterisations

6.1.1.1. First formulation (ψ-s curve)

The particular FD's considered are completely defined by the parameterisation adopted. Invariably it is assumed that the boundary to be transformed is scanned in an anti-clockwise direction and made available in a data structure.

suppose that the simple closed C curve is given the parametric representation:

$$(x(l), y(l)) = c(l)$$

where $0 \leq l \leq L$ is the arc length and L is the full length of the curve C.

If the angular direction of the C at point l is given by $\theta(l)$, then the cumulative angular function defined as: $\Phi(l) = \theta(l) - \theta(0)$ is the net amount of angular bend between starting point l=0 and point l (figure 6.1 (a)). Note that since C is closed Φ is a periodic function of l (with period 2π). Zahn and Roskies [Zahn] further defined the function $\psi(t)$ as follows[1]:

$$\psi(t) = \Phi(Lt/2\pi) - t \qquad (6.1)$$

where t ranges from 0 to 2π.

ψ has very interesting properties, namely, it is invariant under translation, rotation and scaling. Besides, $\psi(t) \equiv 0$ for a circle. Expanding $\psi(t)$ into its Fourier series:

$$\psi(t) = A_0 + \sum_{k=1}^{\infty} A_k \cos(kt - \alpha_k) \qquad (6.2)$$

The set $\{A_k, \alpha_k | k=1,..,\infty\}$ are the FD's of the curve C.

Following are some of the advantages and disadvantages of these FD's. Amongst the advantages :

1. The set $\{A_k, \alpha_k\}$ contains no redundant information.
2. The set $\{A_k, \alpha_k\}$ is invariant under translation, rotation and scaling[2].

---

[1]Note that the formulation of Φ^* here is slightly different because it was decided to scan C anti-clockwise.

[2]The second assertion is in fact a consequence of the first.

Amongst the disadvantages:

1. $\psi(t)$ contains discontinuities when C does; therefore, the A_k have high frequency components and the truncation of the sequence introduces large errors.
2. Reconstruction of C from the A_k is complicated.

6.1.1.2. Second formulation

As reported in [Persoon] a different parameterisation of a closed boundary C can be represented by $(x(l), y(l)) = c(l)$ where l is the arc length along C (anti-clockwise). A point moving along the boundary generates the complex function $u(l) = x(l)+jy(l)$, which is periodic of period L (the length of the boundary). The FD's for such a list are:

$$a_n = \frac{1}{L}\int_0^L u(l)e^{-j\frac{2\pi}{L}nl}dl \qquad (6.3)$$

and

$$u(l) = \sum_{-\infty}^{\infty} a_n e^{jn\frac{2\pi}{L}l} \qquad (6.4)$$

The FD's thus defined present many advantages e.g.:

1. u(l) contains no discontinuities, hence, $|a_n|$ decreases fast as $n\rightarrow\infty$
2. reconstruction of C can be done easily[3]

Among the disadvantages:

1. Because u(l) is a complex function, symmetry is lost i.e. $a_{N-n}^* \neq a_n$
2. The definition of u(l) leads to $\left|\frac{du}{dl}\right| \equiv 1$, this in turn leads to restrictions on the a_n[4] for partial sums.

Alternatively, x(l) and y(l) can be considered as two separate real sequences and the FDs are defined as follows [Kuhl][Chen]:

---

[3]This is not a major advantage since shape recognition is to be done in the frequency domain.

[4]This can be seen by replacing equation (3) for u(l) in the differentiation.

$$x(l) = a_0 + \sum_{n=1}^{N} a_n \cos\frac{2n\pi l}{L} + \sum_{n=1}^{N} b_n \sin\frac{2n\pi l}{L}$$
$$y(l) = c_0 + \sum_{n=1}^{N} c_n \cos\frac{2n\pi l}{L} + \sum_{n=1}^{N} d_n \sin\frac{2n\pi l}{L} \qquad (6.5)$$

6.1.1.3. Normalisation

In this section the normalisation of FD's with respect to starting point is considered for the first formulation given above. Scaling, translation and rotation have no part to play in this normalisation procedure because of the invariance of the FD's (equation 6.1) to these operations. As stated above this is due to the definition containing no redundancy in the definition of a curve C.

Moving the starting point along C is equivalent to a time shift in one dimensional signals. Using the time-shifting property of the Fourier transform it can be seen that the equivalent operation in the frequency domain would be a multiplication the *nth* component by: e^{jnT} where T is the fraction of a period through which the starting point is shifted. It is worth noting that as T goes from 0 to 2π, the starting point moves around the whole boundary once.

In [Persoon] the authors propose a sub-optimal method for the normalisation of FD's. Since they use the second formulation (above), the normalisation includes scaling and rotation as well as shifting the starting point. The method consists of scaling and translating the coefficients in the frequency domain thus making a(0)=0 and |a(1)| = 1; and then applying the combined rotation and starting point shift. The desired Normalised FD will then have a(1) and a(-1) of equal phases. Although the method works quite well on relatively smooth shapes, there are cases where after scaling a(-1) becomes insignificantly small. This leads to problems in the determination of the rotation and starting point shift operator.

Wallace and Wintz [Wallace] suggested that instead of a(1) and a(-1), the two coefficients in the sequence that have the largest magnitude should be chosen for the zero phase condition. It is worth noting that a(1) will always be the largest coefficient in magnitude (fundamental frequency). Therefore, the second largest coefficient has to be found and used with a(1) in the normalisation criterion. However, since the second largest coefficient is not generally a(-1) but a(k) then it can be shown that there are

|k-1| possible orientation/starting point combinations that satisfy the zero phase condition. In [Wallace] the maximisation of a function of the frequency coefficients is proposed for choosing the combination.

The alternative definition of the second formulation of FDs can be normalised as follows:

$$\begin{bmatrix} a_n' & b_n' \\ c_n' & d_n' \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} a_n & b_n \\ c_n & d_n \end{bmatrix} \begin{bmatrix} \cos(n\theta) & -\sin(n\theta) \\ \sin(n\theta) & \cos(n\theta) \end{bmatrix}$$

where

$$\theta = \begin{cases} \theta' & \text{or,} \\ \theta' + \pi \end{cases} \qquad (6.6)$$

and

$$\theta' = \frac{1}{2}\arctan\left(\frac{2(a_1 b_1 + c_1 d_1)}{a_1^2 + c_1^2 - b_1^2 - d_1^2}\right) \qquad 0 \le \theta < \pi$$

$$\begin{bmatrix} a_1^0 & b_1^0 \\ c_1^0 & d_1^0 \end{bmatrix} = \begin{bmatrix} \cos(\theta') & \sin(\theta') \\ -\sin(\theta') & \cos(\theta') \end{bmatrix} \begin{bmatrix} a_1 & c_1 \\ b_1 & d_1 \end{bmatrix} \quad \text{and} \quad \psi = \arctan\left(c_1^0 / a_1^0\right)$$

The effect of the above transformation is to render the two vectors (a'_1, b'_1) and (c'_1, d'_1) perpendicular and then rotate them onto the co-ordinates axis so that $c'_1 = 0$ and $b'_1 = 0$. To complete the normalisation process every coefficient is divided by a'_1.

6.1.1.4. Methods adopted

Fourier descriptors have been used extensively for aircraft and character recognition. The majority of applications reported in the literature use variations of the second formulation (above). In this work both ψ-curve FDs and complex FDs are considered.

6.1.1.4.1. ψ-curve based FDs

ψ-curve based FDs have interesting properties:

1. The spatial coefficients are real, and

2. They are invariant under rotation, scaling and translation.

Because the spatial domain coefficients are real, the Fast Hartley Transform (section 6.1.2) can be used to improve performance. Moreover, the invariance under rotation, scaling and translation simplifies the normalisation procedure, since only normalisation with respect to starting point is needed. The rationale is that before considering a performance increase through parallel implementation, it is worth pondering the possibility of ameliorating the performance of sequential algorithms.

FDs based on ψ-curve present, however, one major problem; and that is the frequency coefficient a(k) will decrease rather slowly when $k \rightarrow \infty$. One way around this problem is to trace the boundary with variable speed, which was first proposed in [Persoon]. Covering the boundary with variable speed will reduce the discontinuities due to corners in the boundary. This can be achieved by taking more samples around the regions of high curvature.

The method proposed here for varying the speed involves scanning the boundary once to associate a curvature measure with each sample in the linked list of edge elements. The curvature measure is not defined explicitly, instead the frequency distribution of ψ is calculated during the first scan [Ballard]. At this stage the perimeter of the boundary is also calculated. This information is then used to determine the intervals between samples during the second scan. Peaks in the frequency distribution represent high curvature regions. Therefore, more samples are taken from these regions. This is done through modulating the intervals between samples with the frequency distribution. In other words, the distance between two samples is proportional to the frequency distribution.

Were the speed uniform, the interval between two sample points would have been constant, I = P/N, where P is the perimeter of the shape i.e. number of pixels on the boundary and N is the fixed number of samples chosen to represent the shape.

6.1.1.4.2. Complex FDs

The alternative definition given in section 6.1.3 above was also selected for implementation. The main reason for this choice is the simple quasi-unambiguous[5] normalisation procedure. Moreover, because the co-ordinates of the points around the

[5]Cases where the normalisation is ambiguous can be handled by the harmonic invariants [Lin.]

boundary are seen as two separate arrays of reals, the Fast Hartley Transform (described next) can replace the FFT for added performance.

This formulation does not require the gradient direction of edge pixels around the boundary. Therefore, the data-structure feeding data to the algorithm is a list obtained from the chain encoding [Duda] of the curve. However, the closed curve is still covered twice, once to evaluate the perimeter and a second time to register the sample points. The boundary, in this case, is traversed at constant speed. Once the transform of the co-ordinate sequences is computed, the normalisation parameters ψ and θ (equations 6.6) are evaluated and the frequency domain coefficients normalised.

6.1.2. The Hartley transform

The Hartley transform (HT) was introduced by R.V.L. Hartley in 1942. It essentially achieves better performance than the Fourier transform in terms of speed and storage resources by disregarding explicit manipulation of the phase information[6]. HT maps a real function of time x(t) onto a real function of frequency H(f); therefore only single arithmetic operations are needed to compute it. Equations 6.7 and 6.8 show the Hartley transform pair:

$$H(f) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)\,\mathrm{cas}(2\pi ft)\,dt \qquad (6.7)$$

$$x(t) = \int_{-\infty}^{\infty} H(h)\,\mathrm{cas}(2\pi ft)\,df \qquad (6.8)$$

where $\mathrm{cas}(2\pi ft) = \cos(2\pi ft)+\sin(2\pi ft)$.

These two equations are very similar to the formulation of the Fourier transform and its inverse. The only difference between the two functions is that the real function cas(2πft) in Hartley transform replaces $e^{\pm j2\pi ft}$ in the Fourier transform. The discrete form of HT is given in equations (6.9) and (6.10):

$$H(f) = \frac{1}{N}\sum_{t=0}^{N-1} x(t)\,\mathrm{cas}(2\pi ft/N) \qquad (6.9)$$

---

[6]The phase information is still present in the frequency domain since simple arithmetic on the Hartley transform can yield the Fourier transform unambiguously.

$$x(t) = \sum_{f=0}^{N-1} H(f)\mathrm{cas}(2\pi ft/N) \qquad (6.10)$$

As presented above the discrete Hartley transform (DHT) suffers from the same problem as the Fourier transform, namely, N^2 arithmetic operations are needed to compute the transform of an N-element data set. However, due to the similarity between the two transforms, the same mechanism that was introduced by Cooley and Tukey in 1965 [Cooley], for the computation of the fast Fourier transform (FFT), can be used to compute what will then become the fast Hartley transform (FHT).

Essentially, the FFT algorithm uses a permutation process to bisect data until data pairs are reached. The idea behind the permutation process is that it is faster to split data into pairs, compute the transforms of the pairs and then recombine these to make the entire transform. The latter still needs a power of two number of points to compute efficiently.

Bracewell [Bracewell] showed that a similar methodology can be employed for the Hartley transform. Through the application of the shift and similarity theorem he derived the following general decomposition formula:

$$H(f) = H_1(f) + H_2(f)\cos(2\pi f/N_s) + H_2(N_s - f)\sin(2\pi f/N_s) \qquad (6.11)$$

Figure 6.2 shows the signal flow graph for the FHT which is a consequence of the decomposition formula.

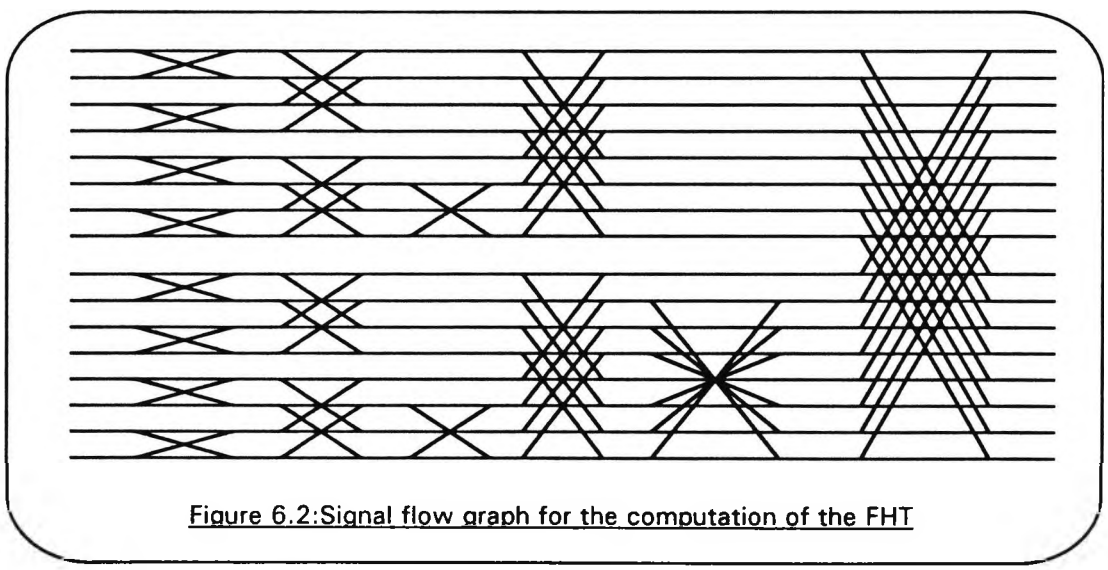

Figure 6.2:Signal flow graph for the computation of the FHT

The Fourier transform can be obtained from the Hartley transform as follows:

$$F_r(i)=H(i) + H(N-i)$$

and

$$F_i(i)=H(i) - H(N-i)$$

Where F_r is the real part of the Fourier transform and F_i is the imaginary part. In fact, it is faster[7] to compute the Fourier transform via the FHT than via the FFT, because in computing the butterfly of Figure 6.2 floating-point numbers are used instead of complex numbers (pairs of floating-point numbers).

The FFT implementation on multi-processors is relatively straightforward due to the inherent parallelism in the butterfly operations.

6.1.3. Implementation

In a typical application of Fourier descriptors a number of curves are stored as FDs, an incoming curve is transformed and compared to the database for the best match. In this work the effort is concentrated on providing fast conversion from images to features (FDs). The problem of searching a large database for the nearest neighbour is quite complex. One approach would be to use a technique similar to that of [Omar 1], where a network of transputers was used to implement a vector quantisation coder/decoder.

In the remainder of this section a transputer implementation of the two FDs mentioned above is described. The algorithm consists of the following:

1. Read an image from the frame grabber
2. Compute edges and edge direction
3. locate closed boundaries
4. Compute FDs and pass onto search routine

In both implementations of FDs the timings and results are obtained for the computation of the Hartley transform, normalisation and in the case of ψ-curves determination of the boundary scanning speed. The curve to be parameterised is

[7]Bearing in mind the additional step for the FFT reconstruction.

assumed to be presented to the system in a standard Freeman chain code (8-connectivity).

6.1.3.1. ψ-curves Fourier descriptors

The main problem of the ψ-curve formulation for FDs is that the magnitude of the frequency domain coefficients does not reduce fast enough as frequency components are added to the sequence. This is due to the inherent discontinuities in the ψ-curve. In other words, a corner in the boundary to be described will give a sharp change in the magnitude of the ψ-curve. Figure 6.3 shows a simple square shape and figure 6.4 and 6.5 illustrate the problem. Figure 6.5 represents the Fourier transform of ψ-curve (figure 6.4) of the simple shape.

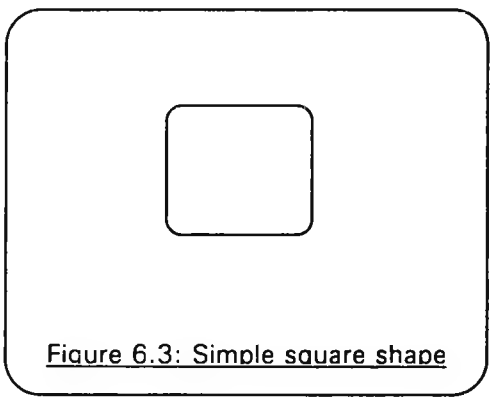

Figure 6.3: Simple square shape

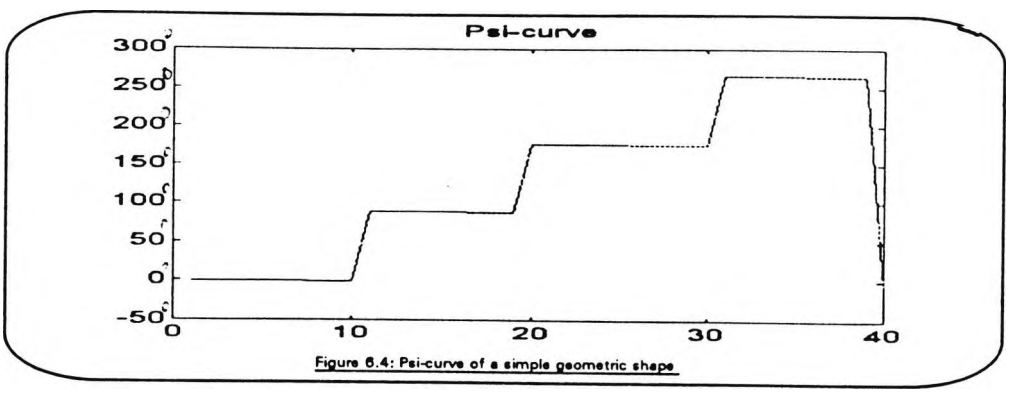

Figure 6.4: Psi-curve of a simple geometric shape

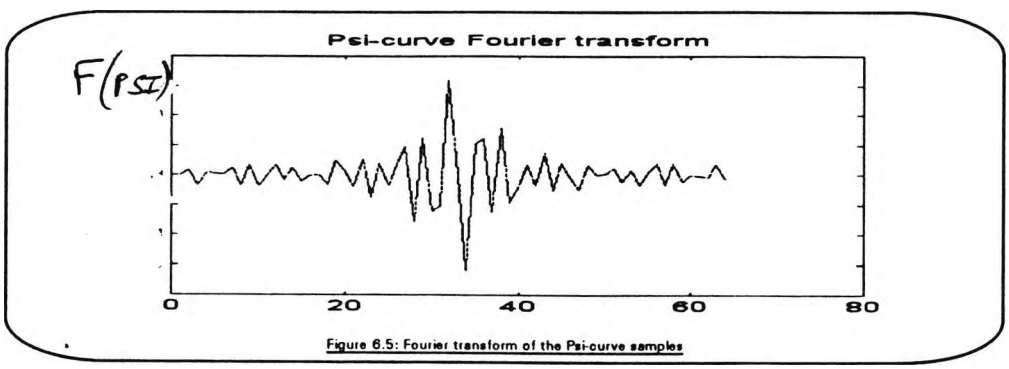

Figure 6.5: Fourier transform of the Psi-curve samples

Consequently these FDs require a large number of coefficients to represent boundaries with some accuracy.

Appendix D-1 shows Occam code fragments which implement the various tasks required to perform edge magnitude and direction evaluation, curvature evaluation, sampling, FHT and normalisation.

The frequency distribution of direction change is obtain by scanning a boundary and representing the instantaneous curvature as the difference between the tangent at the current point and the previous point. Figure 6.6 shows this function for the simple object of figure 6.3

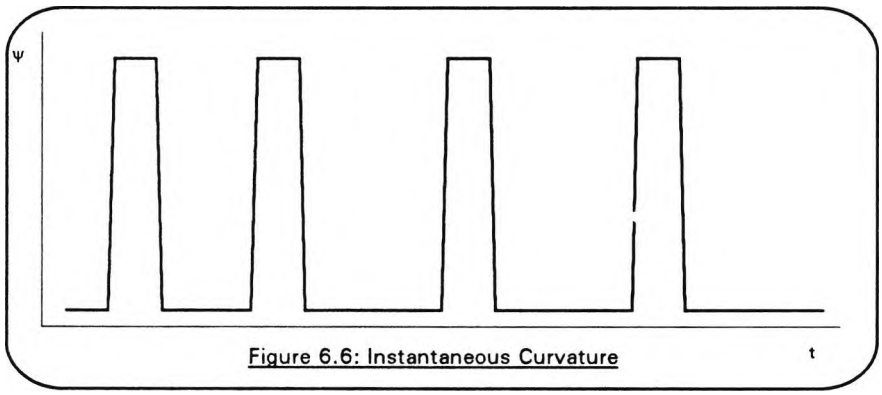

Figure 6.6: Instantaneous Curvature

The curvature modulates the interval between samples. This is done through finding the minimum and maximum curvature in a given shape, then computing the curvature magnitude sampling rate as follows M_S = N/(max-min). Therefore, the distance between a sample and the next is curvature*M_S. Thus, at points of high curvature more samples are taken than at points of low curvature. Although the frequency spectrum of the FDs thus defined is reduced it is still wider than that of the alternative FDs. The experimental set-up is identical to that of the next section.

6.1.3.2. Alternative FDs

In this implementation the boundary is presented to the system as a list of co-ordinate pairs. The main performance increase stems from the parallel implementation of the FHT as a means to compute FDs. The algorithm can be summarised as follows:

1. Parallel FHT on abscissa and ordinate sequences,
2. normalisation through the operations defined in equation 6.6,

3. nearest neighbour classification.

For this third point it is assumed that a library of candidate shapes have been transformed. Figure 6.7 shows the structure of a transputer network to implement the algorithm.

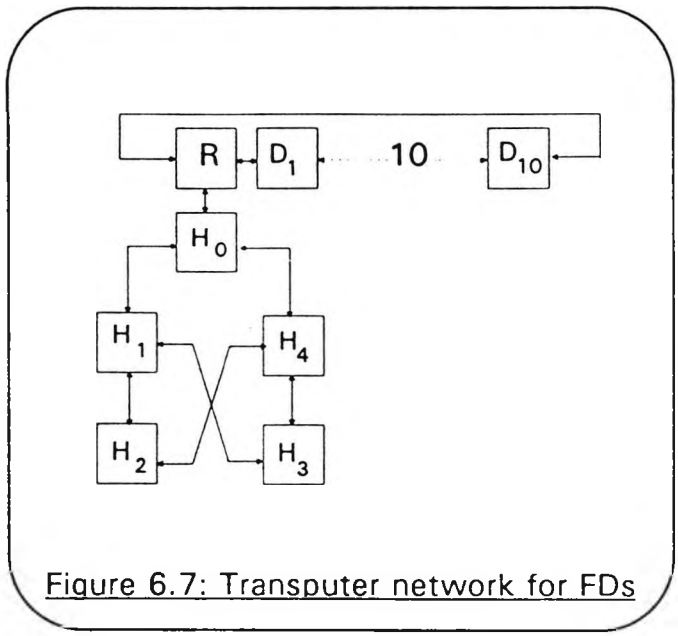

Figure 6.7: Transputer network for FDs

The transputer labelled R is the root processor and resides on a board in an IBM PC compatible computer. It has 4 MBytes of RAM and runs the transputer development system as well as a process to control the network, dispense data and gather results. All the other transputers are TRAMs consisting of a transputer and 1 MByte of RAM housed on a B012 board. The latter together with MMS2 is used to organise the transputer into the configuration of figure 6.7.[8]

The processors labelled H_i perform the Hartley transform, Fourier conversion and normalisation. Their configuration is a direct result of the signal flow graph of the FHT (figure 6.2). A sequence of co-ordinates representing a boundary is passed from the root transputer to H_0 which performs the bit reversal operation. Then, the sequence is partitioned into 4 parts which are passed on to H_i ($1<i\leq4$). Next, these processors perform the FHT on their respective quarter-length sequence. After each transputer has carried out an FHT of quarter-length (H_1,H_2) and (H_3,H_4) communicate to perform half length FHTs. Finally, (H_1,H_3) and (H_2,H_4) communicate to yield the full length

[8]Module Motherboard Software is a software tool allowing the configuration of a network through the C004 switches residing on the B012 board.

FHT. At this point, data is sent to H_0 which derives the Fourier transform. Please refer to figure 6.7bis which represents the signal flow graph of the algorithm.

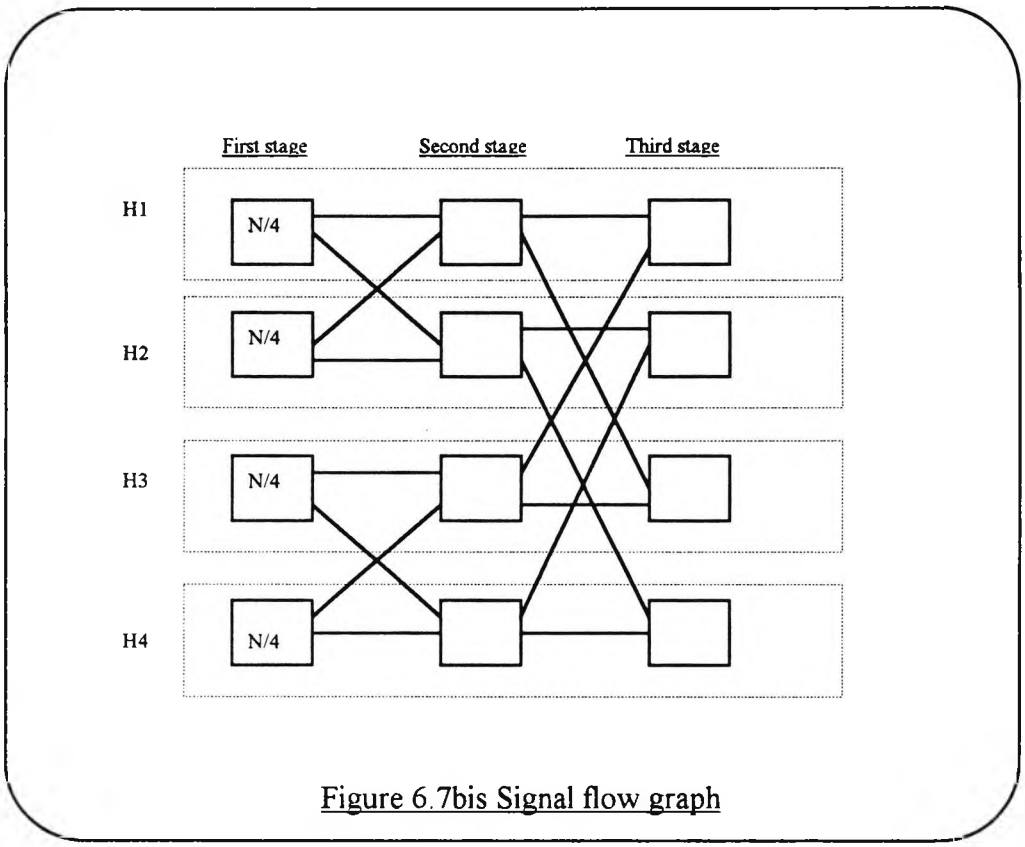

Figure 6.7bis Signal flow graph

Once the two half-sequences have been obtained (at H_0 from H_1 and H_4) the normalisation process begins. Here the parameters θ and ψ (equation 6.6) are evaluated on H_0 and dispatched to the H_i's. Note that this process only requires communications of the type (H_1,H_2) and (H_3,H_4).[9] The full benefit of the transputers labelled D_i $(1<i\leq10)$ can only be felt for large libraries of shapes. Therefore, the timings presented here do not include the nearest neighbour computation[10]. Some simple geometric shapes were used to test the operation of the algorithm. Figure 6.8 shows the boundaries that underwent the process. The number of descriptors required to discriminate between these shapes is very small (8) and might be insufficient for more comprehensive libraries.

[9]Even these transfers are only required because of the data paths.

[10]Since only five shapes were used in the experiment, the classification process becomes trivial.

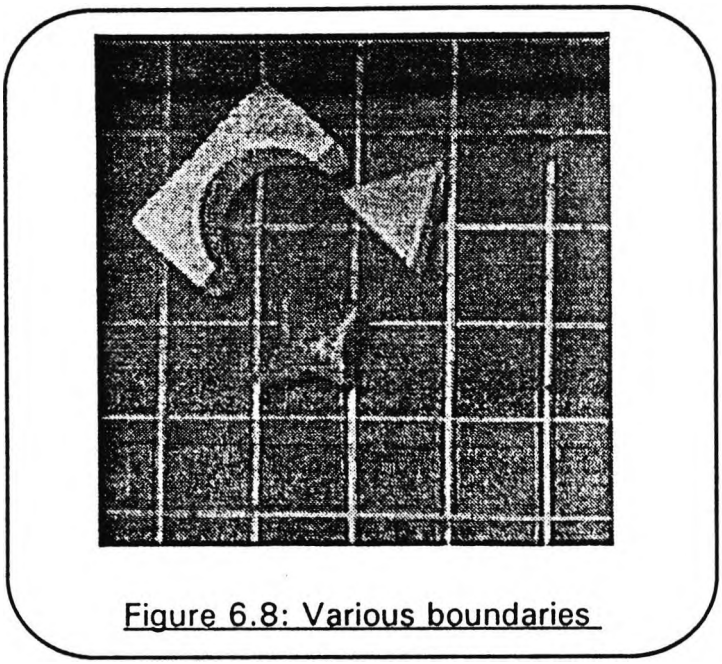

Figure 6.8: Various boundaries

Table 6.1 shows some timings (all times are given in milliseconds[11]) for the whole process (excluding classification). Single precision floating point arithmetic operations are used throughout.

Table 6.1

	1 transputer (ms)	6 transputers (R and H_i in figure 6.7) Network (ms)
128 pixel/boundary	34	13
256 pixel/boundary	78	31

The inefficiency (i.e. the fact that the speedup achieved is not proportional to the number of processors) seen in table 6.1 is due to the lack of symmetry in the signal flow (figure 6.2). The situation will undoubtedly worsen if more transputers were added to the network. In other words, the FD algorithm is not scalable. However, the goal here is to provide a system which provides FDs at a high rate. The fastest chain encoding implemented here (on one transputer) performed the operation in just over 15 ms. Therefore, the system presented here yields high enough throughputs for real applications[12]. Also, the building components are available (no custom hardware

[11]Rounded to the nearest millisecond.

[12]Altough real applications might adopt a method that has a higher throughput with regard to boundary accumulation, they will have to deal with partial occlusions, multiple boundaries and large shape databases

design) and the transputers can be configured without external memory thus making the application commercially viable.

6.2. Parallel Hough transform

Since its inception in 1962 by P.V.C. Hough [Ballard] the Hough Transform (HT) has been used for line, conic and general parameterised curve detection in images. The method is particularly well suited when little is known about the location of a curve but its shape is given in a parametric form. Moreover, it copes very well with noise, gaps and partial occlusions. The basic idea is that a point (x,y) in image space (x-y) can lie on all lines $y = cx + m$; hence, collinear points contribute consistently to the same value of the pair (c,m). Therefore, if a parameter space (c-m) is constructed from image data, maxima will appear at locations corresponding to long segments.

Duda and Hart [Duda 1] considered the problem of locating straight lines in images. The parametrisation of a line is particularly simple since two parameters completely define a line, namely, slope and intercept (Cartesian co-ordinates)[13] or radius and angle (polar co-ordinates). Many authors have shown the Hough transform to be formally equivalent to template matching and matched filtering [Davies 1]. This gives theoretical weight to the method, since matched filtering is optimum with regard to signal-to-noise ratio under white noise conditions [Lynn].

Kimme *et al* [Kimme] adapted the method for the detection of circles and ellipses. In 1981 Ballard [Ballard 1] proposed a formulation that allows for arbitrarily shaped curves. The principle is similar to that of detecting lines and conics, but the accumulation consists of computing the possible loci of reference points.

---

[13]See figure 6.6

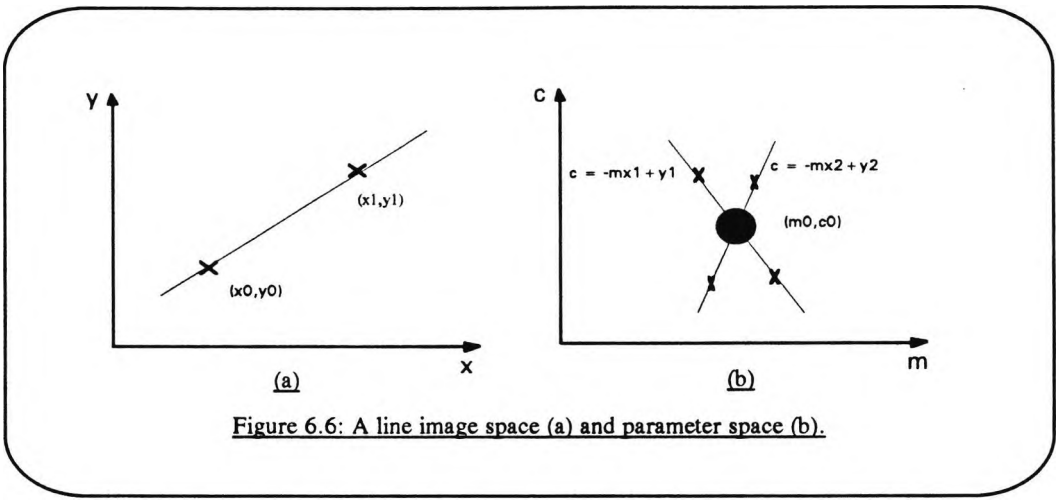

Figure 6.6: A line image space (a) and parameter space (b).

The method thus obtained is a Generalised Hough Transform (GHT). The first step in the GHT algorithm is the detection of edge pixels around the given shape. Each edge pixel then contributes to the parametrisation of the curve as the distance ($R(\theta)$) and angle ($\alpha(\theta)$)[14] from the edge pixel to a reference point inside the perimeter. Having chosen a reference point near the centre of mass of the shape the reference point can be adjusted [Shapiro] to minimise errors due to inaccuracies in the estimation of edge orientation. Then, the values $R(\theta)$ and $\alpha(\theta)$ are put in tabular form. Each edge pixel with orientation θ_1 constrains the set of allowable reference points to $[x+R_1(\theta)\cos[\alpha_1(\theta_1)], x+R_1(\theta)\sin[\alpha_1(\theta_1)]]$. Thus, the table entry for θ_1 contains all pairs (R_1, α_1). Figure 6.7 shows the geometry used to construct the R-table.

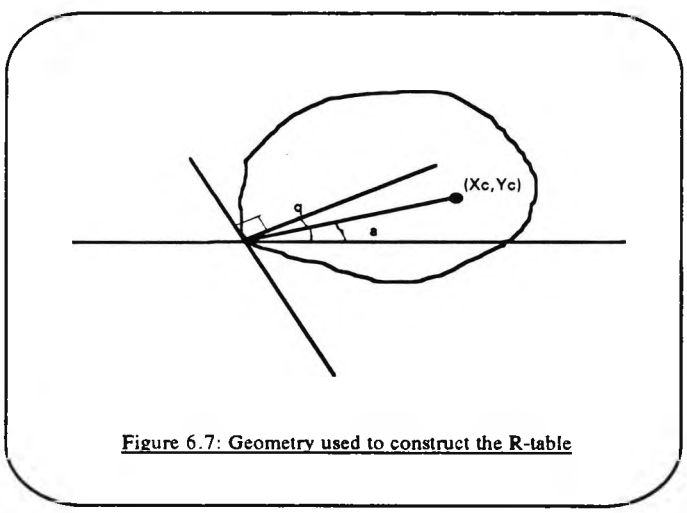

Figure 6.7: Geometry used to construct the R-table

[14]Where $R(\theta)$ is the distance from the edge pixel to the reference point, $\alpha(\theta)$ is the angle from the horizontal at the edge pixel and $R(\theta)$ and θ is the edge orientation.

The R-table can be extended by adding dimensions to the accumulator space to deal with scaling and orientation.

6.2.1. Hough transform for lines

The Hough transform has been studied extensively and its theoretical aspects have been characterised [Davies 1][Brown][Slansky]. It was found to have a sound theoretical basis related to template matching and matched filtering, and relative independence to noise and gaps make the method attractive. However, a major drawback is the very high computational complexity making the method unsuitable for applications requiring real-time (or just high) performance. The next section presents a closer look at the complexity of the HT.

6.2.1.1. Complexity analysis

The parameterisation suggested by Duda and Hart [Duda 1] involves regarding the conversion from polar to Cartesian co-ordinates as a constraint on (ρ,θ) given that (x_i,y_i) $(i = 1, \ldots, N)$ are edge pixels (equation 6.12).

$$\rho = x_i \cos(\theta) + y_i \sin(\theta) \qquad (6.12)$$

Therefore, collinear points in the image space contribute to the same location in a parameter space ρ-θ. The parameter space is then sampled so as to include all possible lines (to the resolution of the image and errors in the edge location and orientation). The ρ-θ space is, hence, an $N_\rho x N_\theta$ array (Accumulator array). The algorithm of Duda and Hart consists of finding the edge pixels, incrementing the loci of points derived from equation 6.12 and finally performing an exhaustive search for maxima in the accumulator array. These maxima correspond to the (ρ,θ) parameter pairs of large collinear subsets of pixels. The ρ-θ accumulator presents a major advantage (over c-m space), in that ρ-θ is bounded. In other words, no line in the image space has $\rho \rightarrow \infty$ or $\theta \rightarrow \infty$. Figure 6.8 shows an image and the corresponding accumulator space. Note that if the gradient angle is computed from the image (at the edge detection stage), a pixel in the image space will contribute to a single location in the parameter space; otherwise, a pixel contributes to a sine locus. Although computing the gradient angle for edge pixels introduces an additional computational load, it leads to a higher signal to noise ratio since noise pixels will have (practically) random orientations.

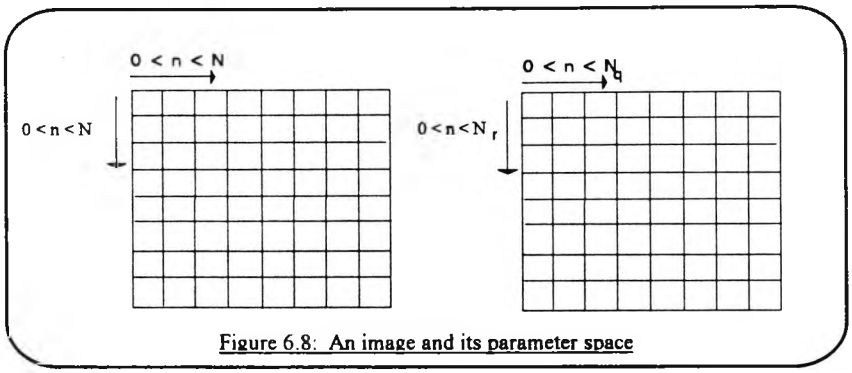

Figure 6.8: An image and its parameter space

The complexity of the edge detection step is ignored in the present analysis[15]. As far as the incrementation stage is concerned, it depends on the number of edge pixels N and N_θ. This is because N edge pixels generate N sets of accumulator space updates; and the cardinality of each is N_θ (by Equation 6.12 and text above). Therefore, the complexity of the incrementation stage is $O(N.N_\theta)$.[16] The exhaustive search for maxima depends on the size (in cells) of the accumulator array; thus, this stage's complexity is $O(N_\rho.N_\theta)$. In general N is quite large (thousands or even tens of thousands for a 512x512 image) and typical values for N_ρ and N_θ are a few hundreds.

Hence, in reducing the complexity of the above algorithm the incrementation stage has to be considered carefully. To this effect, Kiryati *et al* [Kiryati] proposed an algorithm in which the voting for parameter pairs (ρ,θ) is limited to a sub-set of the N edge pixels. They showed that the performance degradation introduced is negligible. The method is based on a probabilistic approach and was termed the Probabilistic Hough transform (PHT). Such reductions in the complexity of an algorithm are limited by the bounds on the number of edge elements in the sub-set and the conformity of the noise statistics in the image to the models adopted. Nevertheless, the performance increase reported is considerable.

A totally different approach to the HT is the Dynamic Combinatorial Hough Transform (DCHT) [Leavers]. The formulation for lines presents fundamental differences with the HT. Firstly, the accumulator array is a 1-dimensional orientation-only space. Secondly, the process iteratively simplifies the input pixels set by removing lines (from the image) corresponding to maxima in the accumulator space. And lastly, because the latter only gives information about direction one pixel is

---

[15]A process like edge detection can with today's technolgy (fast 2-d filtering VLSI components e.g. A110 [InmosSP]) be performed in real time (25 frames/second). One commercially available board the B429 [InmosIQ] consists on two A110's, a T800 transputer, frame buffers and video input.

[16]Appendix B gives the basic definitions of complexity theory (order notations).

selected each iteration, and only lines emanating from it are considered. The technique can be summarised as follows:

0. Initialise the accumulator array: $Acc(\theta_j) = 0$; $0 \leq j \leq$ (Size of Accumulator).
1. Select an edge pixel pix_0.
2. For all other edge pixels pix_i the orientation θ of the line from pix_i to pix_0 increment $Acc(\theta)$.
3. The accumulator space after one iteration will either contain peaks for line segments (of length > threshold) in which case the directions are stored and the lines are removed ; or contain no significant peak and only the selected edge pixel is removed.
4. If no pixels are left terminate, otherwise repeat (go to step 1).

Yuen [Yuen] used this formulation and proposed a method (the Connective Hough Transform (CHT)) based on focusing the search on likely (line) candidates. This is achieved through scanning the image row by row (away from pix_0) and giving no further consideration to orientations (thus pixels) that present gaps (again gap > gap_threshold). The net effect is that edge elements that are not connected do not contribute spurious peaks in the parameter space. Besides the complexity is reduced because of early termination of some iterations.

A major disadvantage is brought about by both the DCHT and CHT. These formulations in trying to solve some inherent problems of the HT, introduce serial operation to the original highly parallel technique. In an SISD model, the above methods would genuinely offer besides the intended effect a reduction of computational complexity - due to early termination, etc. However, the consequence of these techniques on attempts to parallelise the HT will be disastrous.

In the next section a parallel implementation of the HT algorithm is described, and the improvements due to the PHT shown.

6.2.1.2. Parallel Hough transform for line detection

The Hough transform applied to line detection requires the following steps:

1. Initialise the accumulator space ($Acc(\rho,\theta)$) cells to 0,

2. Apply an edge detector (Sobel) to the input image, thus, computing the gradient magnitude g(x,y) and orientation θ(x,y); then threshold the resulting image. For all (x,y) such that g(x,y) > threshold, compute ρ and increment Acc(ρ,θ),

3. Search for the local maxima in Acc(ρ,θ), and keep a number N of them (N depends on the height of the peaks in Acc(ρ,θ).

The Sobel operator can be replaced by a more performant edge detector[17] e.g. Canny. But, the purpose of this section is to parallelise the algorithm and not to improve it. Therefore, the Sobel operator will be used exclusively. It is worth noting, however, that the added processing due to the canny edge detector will not be very taxing, since the complexity of the HT is dominated by the parameter space accumulation.

A close look at the HT algorithm presented above shows that steps 1 and 2 (step 2 excluding the incrementation of Acc(ρ,θ)), are local operations that only rely on a local neighbourhood and are, hence, suitable candidates for the SIMD model. Unfortunately, the accumulation of Acc(ρ,θ) and the search for maxima require global communication. This is because a pixel at location (x,y) can give rise to a wide range of (ρ,θ) pairs; also a local maximum in the parameter space can be insignificant compared to a set of global maxima.

Rosenfeld *et al* [Rosenfeld] report disappointing results on the parallel implementation of the HT on various mesh-connected SIMD architectures:

> *"The analysis presented in this paper shows that a mesh-connected computer composed of bit-serial PEs is not very efficient at implementing an algorithm such as the Hough transform, ..."*

This is certainly true for the types of mesh-connected computers used in their analysis (MPP, GAPP, etc.) These machines are fundamentally different from a mesh-connected computer whose processing element (PE) is a transputer. (The latter does not operate in lock step and synchronises through communication). Firstly, they use the approach termed (Chapter 3) "the army of ants", in other words, the PEs have very limited power and performance is achieved through sheer number, while the transputer

---

[17]Some authors think that the Sobel operator, despite its relative poor perfomance, is better suited for the HT because it yields multiple response to a single edge; therefore, higher peaks in the parameter space.

(of the right generation) is a state-of-the-art micro-processor capable to perform powerful computations. Secondly, these processor arrays are strictly SIMD by construction, whereas a transputer array (or any transputer topology for that matter) can operate both in SIMD and MIMD modes.

6.2.1.3. Implementation

The approach adopted here is the so-called processor farm model (Chapter 8). Because of the data dependencies exposed above (and the fact that processing depends on the number of edges in an image section) a controller processor dictating the operation of the whole network would be unable to balance the load of processing amongst the remaining processors.

In the farm model the controller takes a passive role. As a first attempt a linear farm was implemented. Every processor in the network is connected to two neighbours (apart from the controller and the last worker in the chain). Figure 6.9 shows this simple topology where CTL is the controller and Ws are workers.

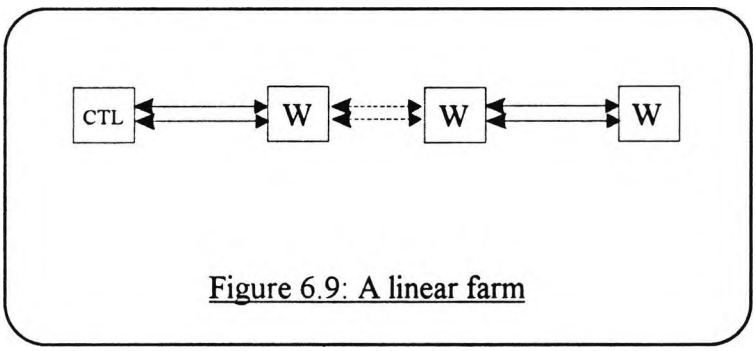

Figure 6.9: A linear farm

Firstly, CTL divides the input image into sections (a multiple of the number of transputers in the network seems a good choice), the limiting factor being the size of the input image. In other words, when there is not enough work no amount of parallelism can achieve better performance.

Secondly, it starts three processes that run in parallel on the controller processor :

1. A Forwarder process (F) that sends the packets formed of a location stamp and an image section,
2. a Load Balancer process (LB) that acts as an interface between F and R, and the rest of the network,

3. and a Receiver (R) process which receives result packets and arranges them into the accumulator array Acc(ρ,θ).

Figure 6.10 shows the controller with the processes represented by ellipses and channels by arrows. This is the case where the controller communicates with the rest of the network through one link only.

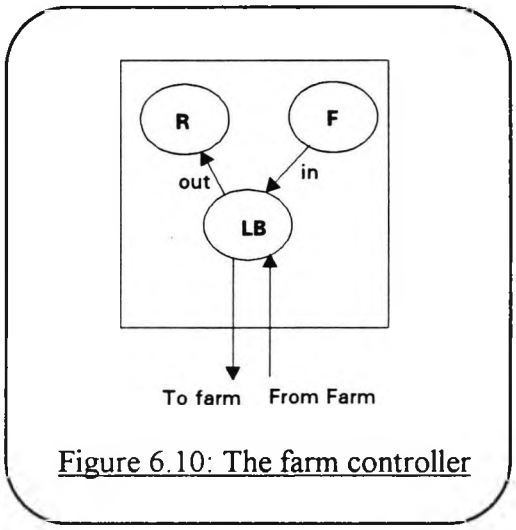

Figure 6.10: The farm controller

The following Occam code segment shows the main actions of the controller:

```
... Declare constants, variables and functions
... initialise variables and read image from frame store
PAR
    Forwarder (in, image)
    Load.Balancer (load.in, load.out, to.farm, from.farm)
    Receiver (out, hough.space)
```

Finally, local maxima detection is performed in the accumulator space Acc(ρ,θ). For simplicity, and because they are not part of the farm side of the application, the image division and maxima detection are omitted from Figure 6.10.

The Worker processors (W) receive an image section, perform the Sobel edge detection, compute (ρ,θ) pairs i.e. entries in the Hough space, and put the latter into packets that are returned to the controller as results. Therefore, workers implement two routing processes as well as the main calculator. Figure 6.11 shows a worker transputer running the following processes in parallel:

1. A Through-Putter process (TP) whose task is to deliver data packets to Ws,
2. a Calculator process (C) implementing the edge detection and parameter space co-ordinates,
3. and a Back-Feeder process (BF) responsible for sending back the results from its local C and other C's further down the chain.

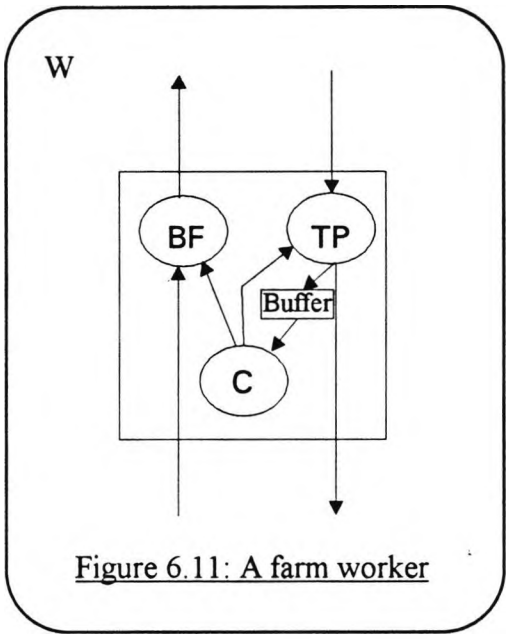

Figure 6.11: A farm worker

The buffer between TP and the calculator ensures that the calculator is provided with data as soon as possible after finishing a sub-task and that the communication blockage is minimum.

To summarise the operation of the algorithm developed in this section let us consider the role of each processor. First the input image is available at the controller which divides it into 33x33[18] patches and farms it out to the workers. (The patch size was arrived at through experimentation. The size chosen gives the best performance in terms of time and load-balancing amongst the workers. Due to the link set up time smaller patches result in greater communication delay, whereas larger patches result in a number of processors remaining idle for longer periods.) The workers for their part get a section perform the Sobel edge detection and pack all the (ρ,θ) pairs into a messages that are sent back to controller. Note that the LB process on the controller keeps track of the number of free processors and only sends packets to the farm if the latter is positive. Finally, the controller performs the peak detection task and files the results. Very marginal improvements on the speedup were noticed when the peak

[18]32x32 sections augmented by a 1-pixel overlap for the 3x3 Sobel edge detection.

detection task is also farmed out. This is due to the fact the operation is less complex and the size of the packets has to be increased in order to overcome the overhead introduced by the division of the Hough space. Also, for a given number of significant peaks corresponding to close parallel lines in the image space, a great deal of computations are duplicated since many of the local peaks are eliminated at the controller.

A concise version of the Occam code for this application can be found in Appendix D-2. The techniques introduced in Chapter 8 for the maximisation of performance were used in the implementation. In short, all routing tasks are run at high priority, and the performance monitor is used to determine a reasonable image section size; although, for a small number of transputers a wide range of values give similar speed-ups.

One major advantage of farming is that a different topology can be implemented with almost (or even no) modification to the workers and very limited alterations to the controller. Figure 6.12 shows a controller which communicates with a farm through two links. F and R remain unchanged but the node manager requires in addition a multiplexer and de-multiplexer process which will keep track of the two branches (free processors and buffers).

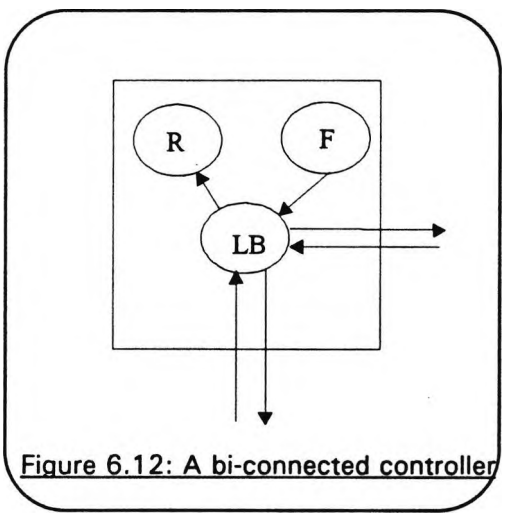

Figure 6.12: A bi-connected controller

Figure 6.13 shows a transputer farm based on the above described controller and workers. The controller resides on a B004 board inside the host computer and the workers are TRAMs sitting on a B012 board in a rack.

The network consists of up to 16 worker transputers forming various basic topologies and a controller communicating with two workers. Next, some experimental results and variations on the structure of the network are presented.

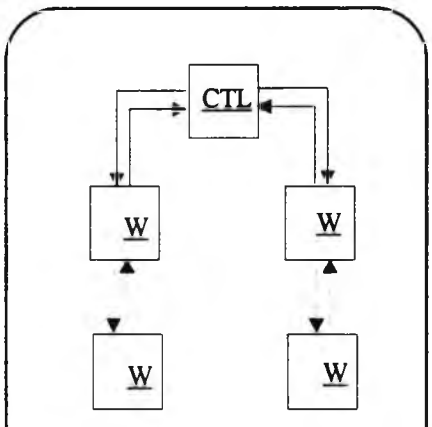

Figure 6.13: Transputer farm

The linear farm and the two-linear farm were implemented for both the Hough transform and the Probabilistic Hough transform. For the latter, the accumulation stage takes place for a subset; namely 20% [Kiryati] of the edge pixels found in the local section of the image. Figure 6.14 shows a graph of the speedup achieved as the number of processors is increased from 1 to 16. The test image is shown in figure 6.15.

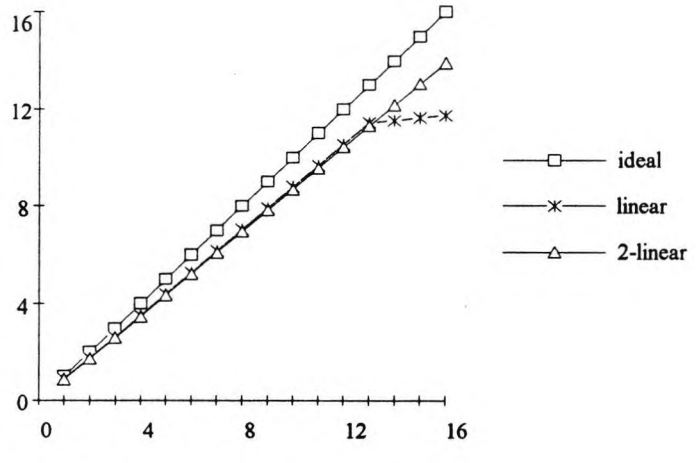

Figure 6.14: Speedup vs. Number of transputers

The speedup achieved can mainly be attributed to the processing to communication ratio. The processors spend most of the time calculating results -as measured by the

performance monitor presented in Chapter 8.[19] In the case of the linear farm the processors halfway through the chain perform slightly better than those at end. This is to be expected since when all the processors are ready to proceed the distance from the controller introduces a precedence relation between them. It is clear from figure 6.14 that the graph flattens when the number of processors exceeds 12 (for the linear farm); the graph for the bi-linear farm is expected to flatten for a number of processors > 16, this is due to the communications saturation. In other words, a delay is introduced between the time a processor finishes work on an image patch and the time it receives the next patch.

Figure 6.15: Test image for the HT

Table 6.2 shows timings for HT and PHT. The difference in complexity between the two algorithms does not affect the speedups achieved, this is due to the similar computation to communication ratios. However, the probabilistic Hough transform as implemented here yields good detection and runs ~3 times faster than the HT on a 16-transputer linear farm.

[19]Besides, each worker kept a count of the number of packets it processed. These values offer more information than the performance monitor (in this particular case.) The reason being that the performance monitor of chapter 5 cannot distinguish between 'useful' computations and communication overheads. The number of sections handled around the middle of the chain was found to be slightly greater than in the vicinity of the controller.

Table 6.2

	1 T	16 T
PHT	2.9s	0.23s
HT	8.5s	0.65s

The edge map of figure 6.15 together with a randomly selected set of edges and their respective Hough spaces are shown in figure 6.16.

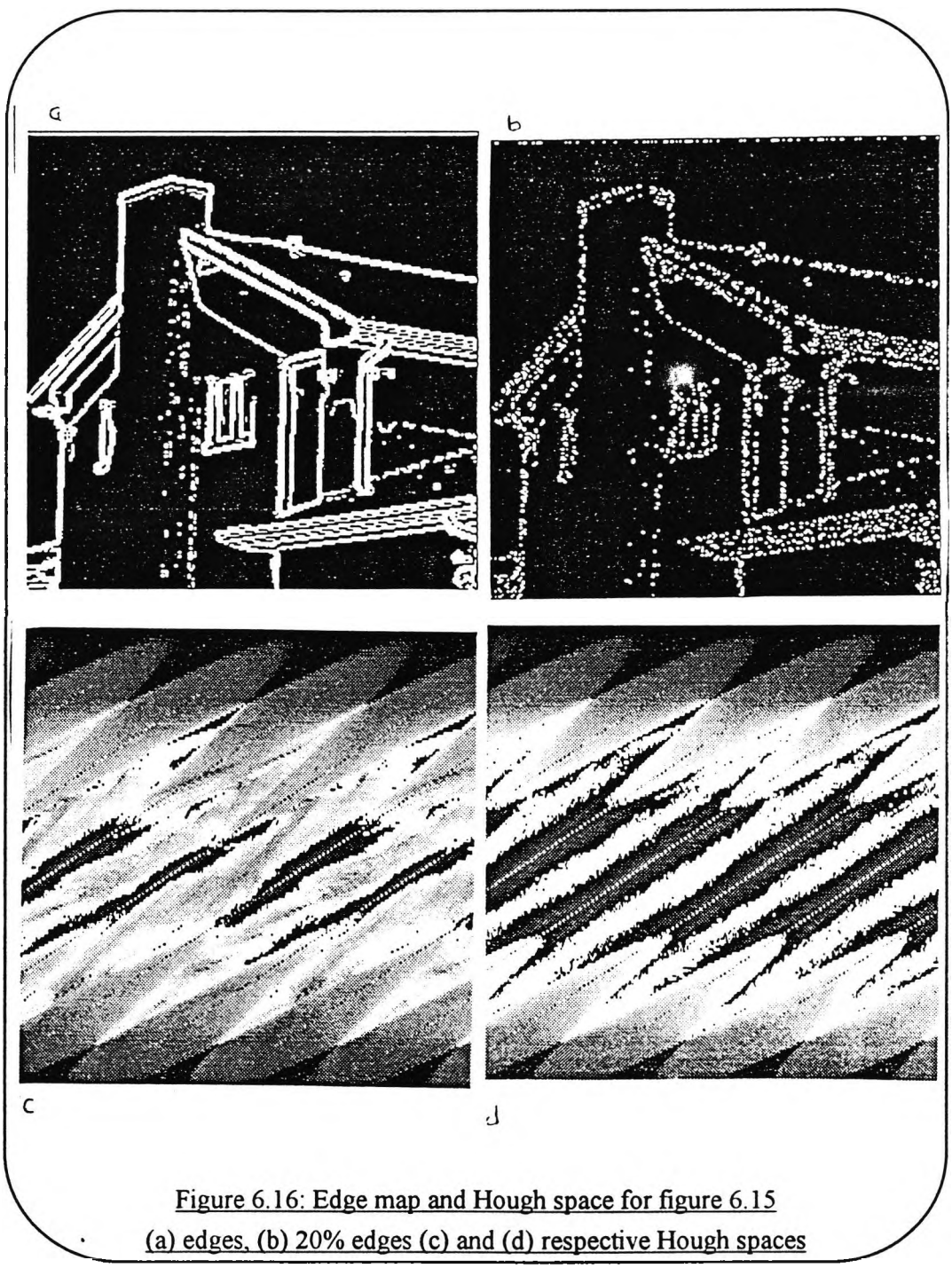

Figure 6.16: Edge map and Hough space for figure 6.15
(a) edges, (b) 20% edges (c) and (d) respective Hough spaces

6.2.2. Generalised Hough transform

A similar set-up was used for the GHT, where an additional step is required in order to compute the R-table. The algorithm is implemented on a linear farm with speedups of up to ~13 on 16 transputers. Since the parameter space incrementation and peak detection are similar to the HT only R-table formation is described here. The algorithm consists of the following steps:

1. Selection of a reference point on the boundary of the shape to parameterise,
2. Compute the distance and rotation of every point on the boundary with regard to the reference point and form the R-table,
3. Form the accumulator space (Hough space) and perform peak detection.

First, an image containing a 'clean' version of the object to be parameterised is read from the frame grabber by the controller. Edge detection at the controller selects a pixel on the boundary to be the reference point (the co-ordinates of this point are broadcast to every transputer in the network) for building the R-table. The latter is computed as follows:

1. The image is divided into sections and farmed out to the processors.
2. In parallel each processor in the network (other than the controller) performs the Sobel edge detection on the local section. For each edge pixel found the magnitude and orientation θ are calculated. Also, the distance R from the reference point is calculated. Then, the angle φ between the horizontal and a line linking the reference point to the edge pixel is computed. The angle φ together with R form an entry into the R-table which is indexed by gradient angle θ. Finally, these entries are packed into a message and sent back to the controller.
3. The controller starts forming the R-table as soon as it receives the first packet from the network. Because some gradient directions will undoubtedly feature more often than others in the R-table, a 1-dimensional array is used to keep track of the number of entries for each allowed direction ($0 < \theta < 180°$).

At this stage the object to be recognised is parameterised in the R-table and the GHT can proceed. A test image is input and its edges are detected and stored in buffers one line at a time. These buffers constitute the packets that are farmed out to the network. Results are returned from the network in the form of (x,y) co-ordinates for the location of a prospective reference point.

Several topologies were tried in order to minimise the distance from the controller to the worker processor. These were linear and bi-linear networks and binary and ternary trees. Very minor improvements if any on the linear farm were noticed. This is due to the relatively small number of processors (16) used in the experiment. Also, partially performing the edge detection on a test image at the controller improves the algorithm's communication requirements and the computation to communication ratio. This idea originates from the fact that the controller does very little between demands from the network for more data. Hence shifting some of the computational load onto the controller results in an improvement in the overall balance. This is particularly true for a sub-task like edge detection whose output contains considerably less data items than its input.

Appendix D-2 gives a shortened version of the Occam code implementing the ideas expounded in this section. Listings 6.1 and 6.2 give the top-level code for the controller and the worker processors respectively.

Listing 6.1

```
...Declarations
SEQ
  ... Initialise arrays
  ... form R.table of object
  ... get new image
 PAR
    Send.data  (to.load, image, scale, angle)
    load.balance  (to.load, from.load, to.farm, from.farm,r.table, performance, buffers)
    Accumulate.Hough.space  (from.load, Accumulator)
```

Listing 6.2

```
... Declarations
... PROC Feed.through (CHAN OF ANY request, to.worker, left.in, down.out, INT pn)
... PROC Worker  (CHAN OF ANY request, to.worker, from.worker, [][]INT results)
... PROC Feed.back (from.worker, soft.channel, left.out, pn)
PRI PAR
  PAR
    Feed.through(request, to.worker, up.in, down.out, processor.id)
    Feed.back(from.worker, down.in, up.out, results, processor.id)
```

```
Worker(request, to.worker, from.worker, results)
:
```

Note that the PRI PAR construct above ensures that the communication processes are run at high priority to avoid blockages. The request channel allows the worker process to ask for more data when it is not busy; therefore, it carries a dummy variable which is discarded at the feed-through process. All the remaining channels implement the same protocol which contains a header tag defining the type of operation, the message destination id, the length and the message. A quasi-linear speedup was achieved by the algorithm running on a network of transputers compared to the sequential version. By no means is this result to indicate that the parallel Hough transform scales linearly. Rather it is a consequence of the high computation to communication ratio which cannot be maintained when more processors are added to the network. The fact that the topology does not seem to affect performance is due to the added complexity of the routing processes when multiple links have to be handled. However, the minimisation of distances between the controller and the workers will become more important for larger farms.

Figure 6.17 shows the image of an object the parameter space (Hough space) displayed as an image (the darker the area the larger the number of votes for that location). The peak is marked with a cross.

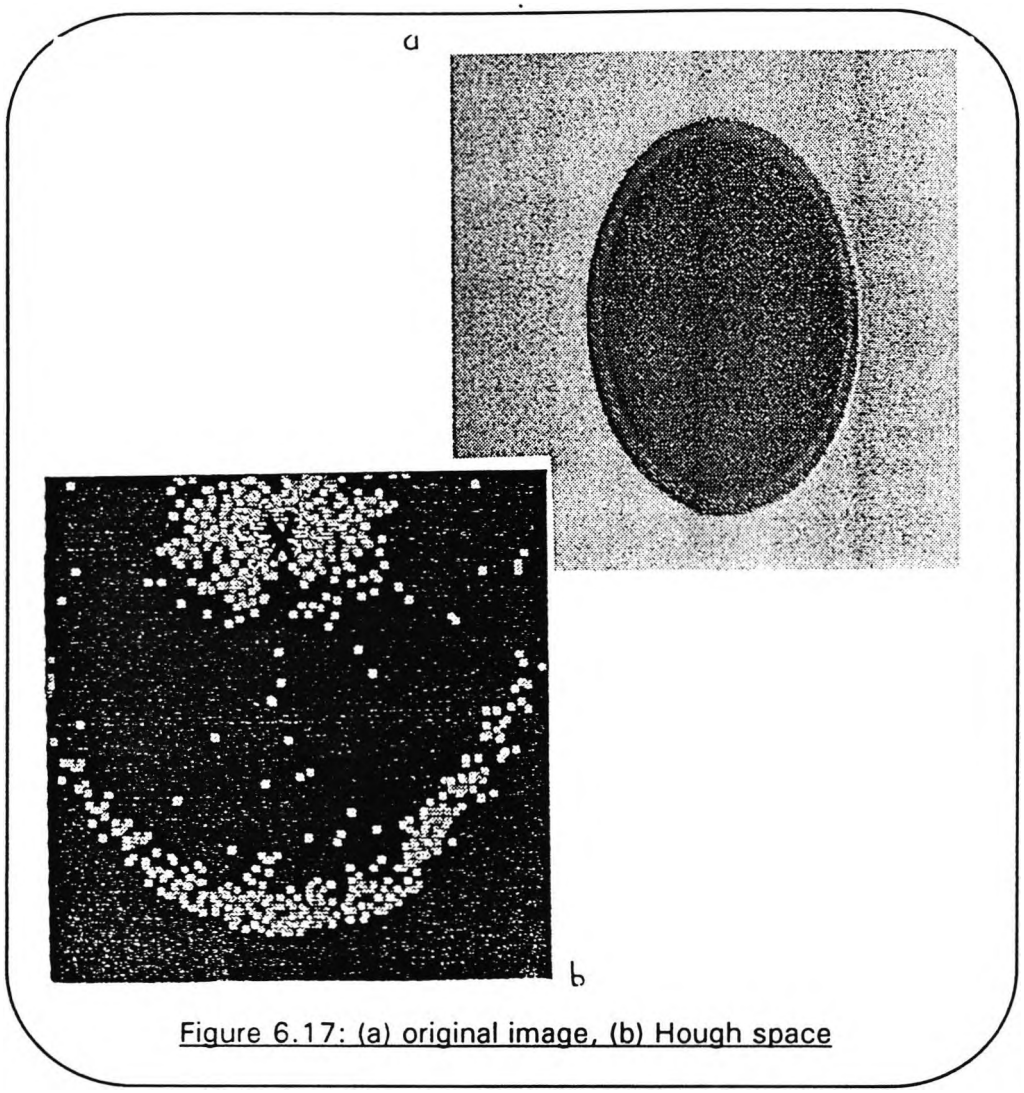

Figure 6.17: (a) original image, (b) Hough space

6.3. Object Recognition using graph matching techniques

In this section the association graph technique is considered. The method departs from the rigour of purely graph theoretical methods such as isomorphism which are too restrictive. The two relational structures to be matched i.e. graphs of the model and the image object are combined into a graph termed the Association Graph (AG) [Ambler]. A node is added to the AG if nodes with same property are present in both relational structures[20]. Edges in the AG are added when two nodes represent compatible relations in the two graphs to be matched. In other words, an edge exists between nodes of the corresponding features in both structures. Therefore, the AG includes information about individual features and their mutual compatibility in the two

[20]Note that the nodes in a graph representation of a scene will correspond to features in the image (or model); therefore, qualifiers (i.e. properties) can be attached to the nodes.

structures to be matched. The largest set of mutually compatible nodes in the AG can thus be deemed to be the 'best match' [Ballard]. This set corresponds to the largest all-connected group of nodes and is called the largest Maximal Clique (MC). An MC is a set of all-connected nodes of a graph that cannot be extended without destroying this property ('all-connectedness'). Therefore, the 'best match' in this method is equivalent to the largest maximal clique.

The remainder of this section is concerned with parallel clique finding algorithms. First, a parallel version of the algorithm proposed by Bolles [Bolles]. Second, the use of transitive orientation as a simplifying procedure is investigated. Finally, the method known as block cluster analysis is studied and implemented.

6.3.1. Maximal clique algorithm

Bolles [Bolles] defines one of the best all-round algorithms for clique finding. The latter is a modified version of the algorithm proposed in [Johnston]. The algorithm relies on the adjacency matrix of the graph under investigation and three sets of nodes:

- The set C is the clique under consideration,
- the set P containing the nodes not in C but which are prospective candidates for inclusion,
- the set L containing nodes that are arbitrarily left out.

The algorithm can be stated as follows:[21]

Maximal_Cliques (C,P,L):-

If (L=Æ) then C is a maximal clique

Else

x **= Choose(L)**

For all $y \in$ **(P** $\cap$ **not(neighbours(**x**)))**

P = P -{y**}**

Maximal_Cliques (C$\cup${y}**, P** $\cap$ **neighbours(**y**), L** $\cap$ **neighbours(**y**))**

A list of all the maximal cliques in a graph is obtained through the call:

---

[21]The procedure Choose selects a member x of L arbitrarily in order to select the neighbours of x in P to extend C. Also, the procedure neighbours(x) returns the elements of V that are adjacent to x and the procedure not(S) takes a subset S of V as argument and returns the complement of S in V.

Maximal_Cliques ({}, V,V) where V is the set of all nodes of the graph under consideration.

Occam does not implement recursion, thus, any attempt at writing a program to carry out the above procedure must start by turning it into an iterative process. This requires the use of a stack data-structure.[22] Work due to J. Arsac [Arsac] establishes the required modifications to the recursive process. In the present simple case of monadic tail recursion (where the recursive call occurs only once as the last instruction of the procedure) the transformation consists of de-coupling the local variables from the body of the task. This is done here through the use of a stack to keep track of the successive values of the variables. The following Occam pseudo-code gives the iterative version of the algorithm presented above.

Listing 6.3

```
BOOL, INT64, INT64, INT64 FUNCTION Max_Cli (C,P,L)
  BYTE x,y:
  BOOL flag:
  VAL
    SEQ
      If
       (L=Æ)
          flag := TRUE      –then C is a maximal clique
       TRUE
          SEQ
            x := Choose(L)
            For all y ∈ (P ∩ not(neighbours(x)))
              P := (P -{y}) ∩ neighbours(y)
              L := L ∩ neighbours(y)
              C := C∪{y}
              Push(L,P,C)
            flag:=FALSE
  RESULT flag,C,P,L
:
```

This procedure can be called to find all the cliques in a graph as follows:

---

[22]As stacks and other data-structures are not available in Occam 2 (with which this work was carried out) it had to be implemented (see [Redfern] for details.)

Listing 6.4

```
BOOL flag:
INT64 C,P,L:
SEQ
  flag, C, P, L := max_cli(∅,V,V) -- Where V is the set of all nodes in the AG
  push(L,P,C)
  WHILE NOT stackempty()
    SEQ
      pop(C,P,L)
      flag, C, P, L := max_cli(C,P,L)
      IF
        (flag=TRUE)
          -- List C as a maximal clique
        TRUE
          SEQ
            flag, C, P, L := max_cli(C,P,L)
            push(L,P,C)
:
```

The above procedure relies on bit-masking and shift operations to select a node from the graph. The node sets C, P and L are implemented as 64-bit integers where the bit position is used as an index in the adjacency matrix to test the existence of an edge. This limits the size of the AG to 64 nodes and is due to the fact that Occam procedures cannot return non-scalar variables (e.g. arrays or sets). To remedy this problem pointers can be passed to global variables containing the node sets.

The method chosen for parallelising this algorithm consisted of the following steps:

- Set up a farm as described in section 6.2 with every worker running the full algorithm with a local stack and return maximal cliques when they are found.
- The controller runs the initial phase of max_cli to fill the main stack and then assumes a passive role whereby it pops values and supplies them to the workers on demand.

The system (based on 9 transputers arranged in a linear farm) was tested on a range of random undirected graphs on 64 nodes. Speedups from ~3.2 on sparsely connected graphs to ~5.4 on densely connected graphs were noted. However, the computation

time for densely connected graphs is significantly larger than that of sparse graphs due to the combinatorial explosion. The latter is a result of the slow decrease of the size of the set L. It is believed that the use of more transputers (and the extension of the algorithm to handle larger graphs) would not lead to significant performance increase because the clique finding algorithm as presented is known to be NP-complete (see appendix B for details on NP-completeness). Therefore, speedups can theoretically only be achieved through the use of an exponentially increasing number of processors. The reason is that as the size of the graph increases the complexity of the problem of finding cliques increases exponentially leading to the need to increase the number of processors accordingly. Also, if the size of the graph does not increase, a greater number of processors will introduce more communications overheads. The following two sections deal with improving the sequential algorithm and parallel versions.

6.3.2. Reducing the combinatorial explosion

This section is concerned with the main cause of inefficiency in the algorithm described above, namely, the arbitrary choice of the element of L which directs the selection of the prospects for clique extension. A pre-processing step is therefore added to the process. The latter consists of generating the map of shortest paths between all pairs of vertices in the graph. This map is then used to select the element of L which is the furthest away from the clique under investigation. The net effect will be a much faster reduction of the set L, thus less time wasted on dead ends. The most efficient algorithm to form this map is given in [Floyd] (see Swamy for details).

Another way to reduce the complexity of the maximal clique algorithm is to apply transitive orientation to the initial graph. The resulting oriented graph simplifies the algorithm [Liu]. An efficient algorithm to this effect was introduced by Pnueli *et al* [Pnueli]. The algorithm was tested on arbitrary graphs and it was found that very often the graph is not transitively orientable. One possible avenue of research into this subject could consider weighing the edges of the AG in order to disregard offending edges during the transitive orientation phase.

6.3.3. Block cluster analysis

This section presents the use of the Block Cluster Analysis (BCA) method to simplify the maximal clique algorithm. The approach relies on the following assertion:

- If the adjacency matrix is re-ordered in such a way that the '1' entries are lumped around the leading diagonal, then it can be partitioned into blocks on which the maximal clique algorithm is run.

The method for BCA follows closely the Ford algorithm. However, the original algorithm is concerned with multi-graphs. Here the intention is to derive an algorithm that will arrange the non-zero entries in the adjacency matrix of a graph nearest the leading diagonal. This can be achieved by minimising the sum of distances of the '1' entries from the diagonal. Let the distance Δ of a matrix entry be measured by $\Delta(A_{ij})=(i-j)^2$. The magnitude of f(A) defined in equation 6.13 decreases when the matrix is reordered so that the '1' entries are closer to the leading diagonal.

$$f(A)=\sum_{i=1}^{N}\sum_{j=1}^{N}A_{ij}*(i-j)^2 \qquad 6.13$$

Therefore, the difference between the magnitude of f(A) for two arrangements of the rows and columns a matrix gives information about the positions of the '1' entries with regard to the leading diagonal.

The search procedure relies on the ordering of the vector representation of the rows and columns of the adjacency matrix A. Let the initial ordering be O_0=(1,2,...n,m,...N) where N is the number of nodes in the graph. Then a different ordering O_1=(1,2,...m,n,...N) can be obtained by swapping the ith row with the jth row and the corresponding columns (to keep the matrix symmetric). Finally, $f(A_1)$[23] is calculated and compared with $f(A_0)$, if $f(A_1) < f(A_0)$ then the swap represents a step towards the optimal ordering. Expanding $f(A_1)$ to separate the entries involving indices m and n from the rest an taking the difference leads to equation 6.14.

$$f(A_1)-f(A_0)=2(m-n)\sum_{\substack{i=1\\ i\neq m\\ i\neq n}}^{N}(A_{im}-A_{in}).(2i-m-n) \qquad 6.14$$

This equation is far less computationally demanding than equation 6.13 since because of symmetry it only involves the elements of the two rows (columns) being swapped.

[23] A_i is matrix A under the ith reoreding

However, to put matrix A into its canonical form, where f(A) is minimal over all permutations of indices, is a very complex task which is believed to be NP-complete. The main idea in this work is to ensure that the choice of elements to swap is effected along the maximum gradient of f(A).

Every processor in the parallel implementation is allocated a set of nodes for which it evaluates the function over all possible swaps. The most negative $f(A_{i+1})$ for each node in the set locates the row (column) with which it must be swapped.

The application showed no speedup due to the counter productive effect of conducting swaps in parallel. The definition of 'best swap' for a particular row (column) is destroyed by other concurrent swaps.

6.4. Summary

This chapter showed that considerable speedups can be achieved on intermediate level computer vision algorithms through the use of parallel processing. However, the better serial algorithms are not necessarily well suited for parallel implementation. Improvements to the serial algorithms that do not affect parallelisation were presented. As far as high level computer vision is concerned as exemplified by the maximal clique algorithm, the speedups were more modest due to the complexity and global nature of the computations involved.

CHAPTER VII

7. System architecture

The aim of this chapter is the design of an interconnection network suitable for a parallel computer vision system based on the transputer (T800). The desirable characteristics of networks are given and translated into requirements on the underlying graph.

A transputer network can be viewed as a graph with the transputers as nodes (or vertices[1]) and the links (bi-directional) as undirected edges. In the study of static topologies, and because the transputer has four links (fixed), only graphs with at most four edges per node need be considered. In order to use all the bandwidth afforded by the transputer links, a network can be constructed each of whose processors is connected to exactly four neighbours. In the terminology of graph theory (section 7.1) this is called a regular graph as all vertices have the same number of edges.

The network proposed here consists of a two-level hierarchy with the nodes connected into groups of eight transputers and the groups, in turn, linked to form the overall network. Because of the simple construction of both levels, routing mechanisms to convey messages between pairs of nodes is simplified. Chapter 8 presents an Occam program that implements routing along the shortest path between any two processors in the network. Also, dynamic configuration is considered, and a scheme is presented which is a consequence of the topology of the groups mentioned above (section 7.3.4).

The remainder of this chapter is organised as follows. First, the terminology of graph theory is introduced. Second, some well known topologies are presented and analysed

[1]Here the terms node and vertex are used interchangeably.

with regard to feasibility (implementation based on the transputer). Then a topology is proposed and assessed. Finally, some concluding remarks are given.

7.1. Terminology

A graph is defined as a collection of vertices some of which are connected by edges or arcs. It can also be thought of as a binary operation on the set of vertices. The latter representation allows for the extension of useful concepts from group theory to graphs. Graphs can be directed or undirected, an undirected graph is, in fact, a symmetric binary relation. All the graphs considered in this work are simple[2]. In other words, they are linear i.e. only one edge is allowed between two vertices (two in opposite directions for the directed case) and they contain no slings (or edges from a vertex to itself)[3].

The order of a graph G(V,E) where V is the set of vertices and E is the set of edges is the number of elements in V. A set of edges in a graph is said to be adjacent in G if they have one vertex in common. Two nodes n_i and n_j are adjacent if there is an edge in E incident on both of them.

A path in a graph is an alternating sequence of vertices and edges starting and finishing with vertices such that: $v_0,e_1,v_1,e_2,.....,v_{k-1},e_k,v_k$ where e_i is incident on both v_{i-1} and v_i $1 \leq i \leq k$. A path can also be considered as an ordered list of vertices in which every pair corresponds to an edge. A graph is connected if there is a path between any pair of vertices in V. Otherwise the number of components is >1 and consists of the number of connected sub-graphs.

A path whose initial and final edge are equal is called a circuit. The number of edges in a path is the length of the path. The shortest path between two vertices is named a geodesic. The diameter of a graph is defined as the shortest geodesic over all pairs of vertices.

The edge connectivity of a graph is the minimal set of edges whose removal will render the graph disconnected. Likewise, the vertex connectivity is the minimal set of vertices whose removal renders the graph disconnected.

---

[2]Exceptions will be noted in the text.

[3]The binary relation is, therefore, not reflexive.

The girth of a graph is defined as the minimal length over all the circuits of the graph.

A graph is a convenient way to represent many problems. Applications of graph theory range from the solution of combinatorial problems through the design of communication networks to the solution of sets of algebraic equations [Temperly][Swamy][Ore].

The degree of a vertex is the number of edges incident on it. A graph is termed regular if all its vertices have the same degree. For the purpose of this chapter the simple definitions above are sufficient. Further notions can be found in chapter 6.

7.2. Network Topologies

The topology of a parallel computer refers to the way different processing units are connected. Many topologies have been proposed. Researchers have been arguing the adequacy of a particular scheme for the implementation of real systems. The outcome is that some inter-connection schemes are superior to others for some applications and paradigms; therefore, the particular topology adopted is very much dependent on the computation and communication requirement of the problem at hand. To assess the value of a topology for image processing and computer vision as a whole would be a very complex task indeed. This is due to the varying degree of communication and optimal grain exhibited by algorithms pertaining to different levels in the hierarchy of a vision system (chapter 2).

Therefore, there is a need for a framework allowing the quantification of the properties of topologies. Such a framework does exist and can be stated as a compromise between a few parameters of the graph representation of the topology. These are the diameter, connectivity and average distance[4]. Another important aspect of a topology is the presence or absence of symmetry and embedability of other architectures. The latter relates to the ability to emulate other (simpler) structures i.e. contain interesting partial graphs e.g. trees etc.

In this section a review of the different topologies introduced in the literature is given together with their parameters. Then, a topology based on a mesh inter-connection scheme is proposed and analysed.

---

[4]See section 7.1 for the terminology of graph theory

7.2.1. Mesh

A mesh is a topology where the nodes are arranged in an *m*-dimensional lattice. Communication is only allowed between neighbouring nodes. Internal nodes communicate with $2m$ nodes. In order to use the connections from boundary nodes, some variants of the mesh include wrap-around connections between processors on the edge of the mesh. Figure 7.1 shows a two dimensional mesh (a) and two variants, same row and column wrap-around connections (b) and toroidal connection (c).

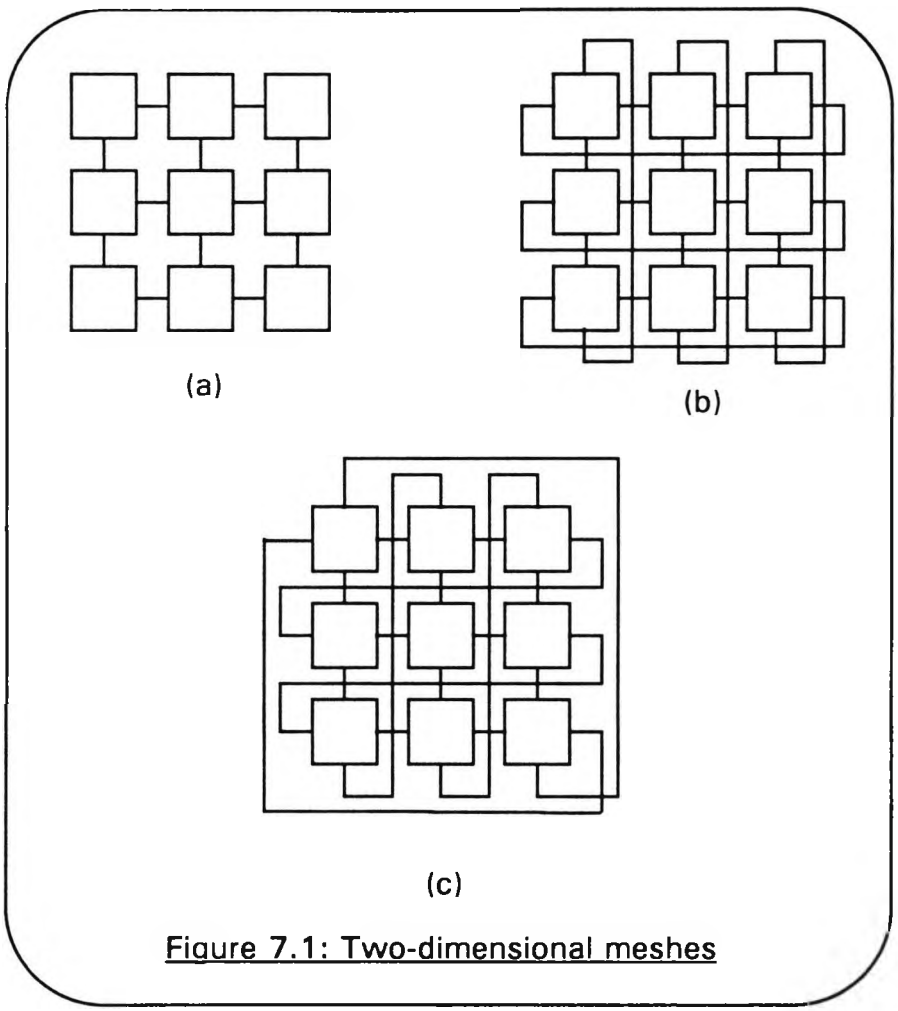

Figure 7.1: Two-dimensional meshes

7.2.2. Pyramid

A pyramid is a complete 4-ary tree augmented with inter-processor links so that each tree level forms a two-dimensional mesh [Quinn]. The Pyramid can prove very useful if the application presents a hierarchical structure with low level data parallelism transforming data into progressively more centralised functional parallelism. However, in the context of transputer networks the maximum degree of nine for internal nodes makes it difficult to implement even through a switching network (crossbar).

7.2.3. Butterfly

A butterfly network is a topology with $(k+1).2^k$ nodes arranged into k+1 columns. Each column contains 2^k nodes. Figure 7.2 shows a butterfly network with 12 nodes (k=2).

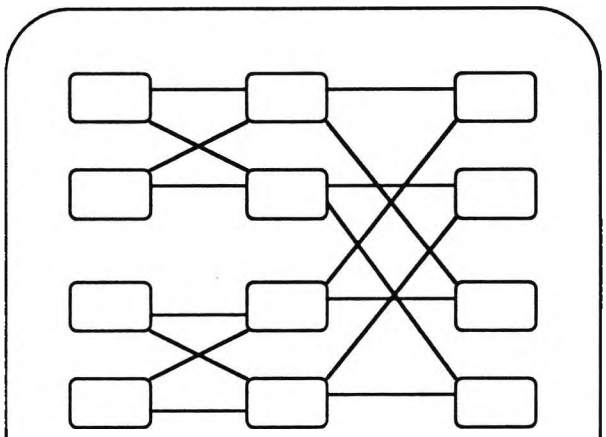

Figure 7.2: A butterfly network with N = 12

The butterfly network is an obvious choice for algorithms like the Fast Fourier Transform (FFT) [Cooley]. The latter, has a precedence graph identical to the butterfly network. Mazzeo *et al* [Mazzeo] describe an implementation of a one-dimensional and two-dimensional FFT on a butterfly based transputer architecture. Batcher [Quinn] showed that a butterfly can perform bitonic sort on n elements very efficiently when n is a power of 2. It is worth noting that the bitonic sort algorithm also has a butterfly precedence graph.

One advantage of the butterfly network is that it is regular i.e. the number of links at each node is constant (=4). However, for large networks, the problem of gathering the results of a computation becomes apparent. This is due to the exponential increase of the volume of data passed between processors when a number of cells are allocated to each processor.

7.2.4. N-cube

This topology, also called a hypercube, has many interesting properties. It can be obtained from the butterfly by collapsing the lines into a single vertex. A d-dimensional hypercube is built with $n = 2^d$ nodes, where each node has degree d. Such a network

will contain $2^{d-1}*d$ edges. A systematic construction rule can be established by noting that the hypercube belongs to a class of graphs known as the graphs on alphabets. These graphs present global knowledge about the location of a particular node. Therefore, routing, which is a very important aspect of parallel processing, can be defined explicitly. Appendix C gives various definitions including that of graphs on alphabets.

In a hypercube each node is connected to a node in each of the *d* dimensions; moreover, the network is symmetric and regular. A building rule can be stated as follows: each node is associated with a binary number, and the number of bits necessary to represent all the nodes is $\log_2(n) = d$. A node $a_1,...,a_i,...,a_d$, is therefore, connected to all nodes whose binary representation is obtained by inverting exactly one of the *d* bits e.g. $a_1, a_2, \cdots, a_{i-1}, \bar{a}_i, a_{i+1}, \cdots, a_d$.

One major advantage of the hypercube is that a particularly simple routing mechanism can be established. The latter is optimal in the sense that messages are sent along shortest paths. A message is transferred from node n_i to node n_j through a set of nodes that reduce the Hamming distance[5] by one at each step. The distance between two nodes is their Hamming distance. Therefore, the largest shortest path (or geodesic) is equal to *d* the diameter of the network. It is worth noting that there are *m*! (factorial *m*) paths between two nodes which vary in *m* positions, *m* of these shortest paths are edge and node disjoint. This has an important bearing on the reliability and fault tolerance of the network.

Moreover, the *d*-dimensional hypercube is a familiar structure[6] that can be subdivided into two (*d-1*)-dimensional hypercubes by setting one bit of the binary representation of its nodes to '0' to obtain the first sub-graph, then to '1' to obtain the second. Note that this is a done across the dimension of the chosen bit position. Such properties are very useful in the context of multiprocessing, if only for the simplicity of allocation of resources (i.e. processors) to processes according to their needs. The dimension of the sub-hypercube allocated to a particular 'independent' process can reflect its processing requirements.

---

[5]The Hamming distance between two binary strings (with same length) is the number of places where they differ.

[6]A 2-dimensional hypercube is a square and a 3-dimensional hypercube is a cube.

However, two problems arise when the implementation of this topology is considered. First, because of the exponential growth in the number of edges (links) required to connect the nodes, a standard node (e.g. processor in a parallel computer) must have a large number of links. This poses serious difficulty, if only on the number of pins. Second, a hypercube exists only for numbers that are powers of two. In other words, to upgrade a system on $n = 2^d$ nodes gracefully (i.e. without major modifications to system software viz. routers, schedulers, etc.) an additional $n = 2^d$ have to be added. These are very serious problems that the Cube Connected Cycles topology tries to remedy.

7.2.5. Cube connected cycles

A cube connected cycles network (CCC) is a *d*-dimensional hypercube where the nodes are replaced by cycles of *d* nodes (figure 7.3). The basic idea behind CCC networks is to counter-balance the exponential growth of the number of edges needed by introducing 'super-nodes' that emulate the necessary increase in the degree. A CCC retains some of the properties of the hypercube like symmetry while increasing the length of the diameter of the network. However, it does deal with the problem of scaling mentioned above, but it further restricts the natural numbers of nodes for which a CCC exists. A CCC only exists on numbers $n = 2^d * d$ nodes.

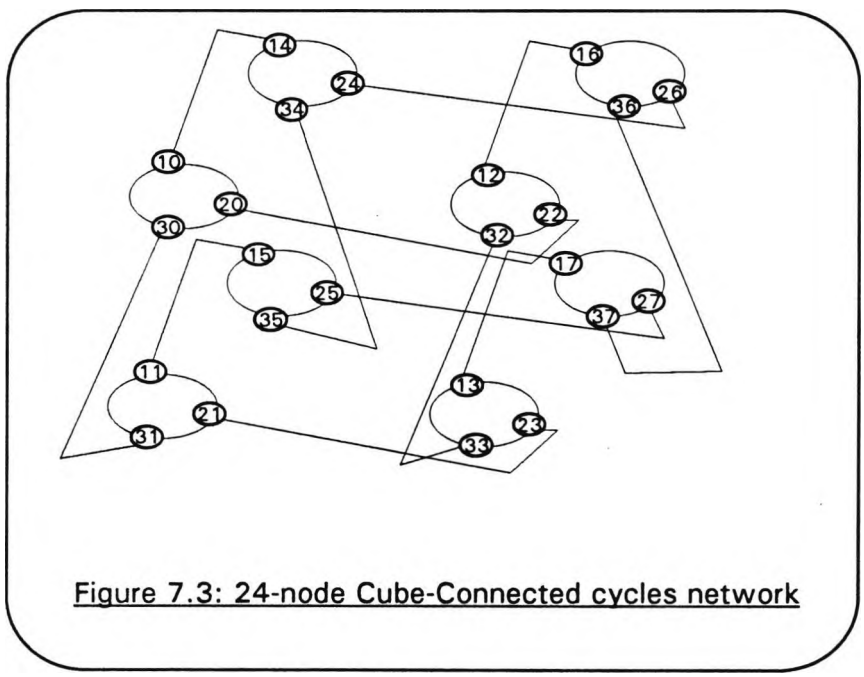

Figure 7.3: 24-node Cube-Connected cycles network

7.2.6. Perfect shuffle

The perfect shuffle network is a special case of a general class called shuffle-exchange networks. These also have a restriction on the number of nodes n, $n = 2^d$. Two types of connections are identified: exchange connections and shuffle connections. Exchange connections link pair of nodes whose binary representation differ on their least significant bit. The exchange connections define the network. For the perfect shuffle network, node i is connected to node $2*i$ modulo n-1. The perfect shuffle is so called because it introduces a lower bound on the maximum number of shuffle operations that return a datum to the node which issued it, namely d.

7.3. Architecture for large transputer networks

A number of authors have studied graphs with given properties and associated graph theoretical concepts with practical considerations in applications [Wilkov]. The advance in technology and the advent of parallel computers revived interest in the diameter minimisation problem with constraints on connectivity (vertex and arc) [Boesch].

As stated above the connectivity is defined as the minimum number of vertices/arcs that have to be removed for the graph to become disconnected. In a parallel computer a disconnected set of processors would be useless. Therefore, an architecture with a large node and link connectivities would be more reliable, since more nodes or arcs have to fail before there is a considerable (or even total) loss of performance.

The diameter of a graph together with the number of edge disjoint paths between any two nodes also affect the reliability. Moreover, the diameter affects communication delays. Therefore, supposing that a distributed program is implemented on two parallel computers (with similar processing nodes) and requires global communication, then the computer with the smaller diameter will perform better. The condition of global communication is required for the above assertion because when the structure and volume of data transfer is known deterministically, a specific topology (possibly with poor diameter, connectivity, etc. properties) would do a better job. However, in the design of a general purpose machine these parameters together with the embedability and regularity criteria are good measures of feasibility, performance and reliability .

Before presenting the class of regular graphs that will be the basis of the proposed architecture, let us consider a graph proposed by De Bruijn [De Bruijn]. De Bruijn

graphs DB(d,N) are directed graphs defined on a set of d^N nodes. The nodes are labelled by the words of length N on an alphabet of d symbols. There is an edge from node *i* to node *j* if the first N-1 symbols of *j* are equal to the last N-1 symbols of *i*. Therefore, an edge links $(i_0,i_1,...,i_{N-1})$ to all nodes of the form $(i_1,...,i_{N-1},j)$, where *j* is any of the *d* symbols of the alphabet. Thus, each node is the starting point of *d* edges and the end point of *d* edges. The diameter of a DB(d,N) graph is equal to N. Figure 7.4 shows DB(2,3).

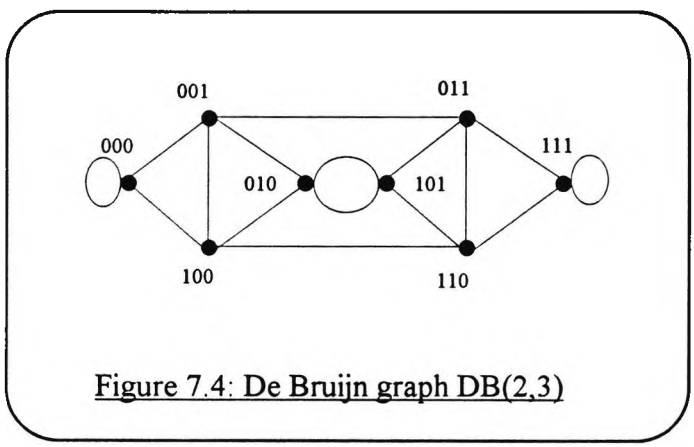

Figure 7.4: De Bruijn graph DB(2,3)

De Bruijn graphs are not regular and neither are they linear if edge direction is discarded. However, they have very impressive diameter properties. Because, like for the hypercube, De Bruijn graphs' node degree is proportional to *d*, only small values of *d* can be considered for interconnection networks. For the transputer[7], with four links only DB(2,N) are realisable without switching due to the construction procedure of DB(M,N) (see section 8.2) This restriction is not very taxing since the diameter properties are compromised. For example, a 1024 node De Bruijn graph will have a diameter of 10 and a maximum degree of 4. Routing through DB(d,N) can be achieved by a left shifting operation defined on the words of length N. Although this scheme is not optimal it provides very good average distances. The simple building rule of DB(2,N) makes these graphs good candidates for the construction of transputer networks.

Another related class of networks known as Kautz graphs present similar diameters and building rules [Baude].

However, the irregular structure of these graphs does not guarantee the efficient or easy mapping of a large class of algorithms that present regular or hierarchical

[7]Appendix A describes the transputer family of processors and the Occam language.

communication patterns e.g. 2-d filtering, FFT, bitonic sort, etc. Another disadvantage of DB(2,N) is the fact that it only exists for specific numbers of nodes, $N_0=2^N$.

7.3.1. Metrics and bounds

Different performance metrics have been proposed for the characterisation of candidate architectures [Wilkov 1][Wilkov][Boesh]. Edge and node connectivity have been studied extensively together with diameter and average distance minimisation. Boesch and Thomas [Boesh] have extended the notion of connectivity to define the cohesion of a regular graph. The cohesion of a graph $\delta(i)$ is defined as the minimum number of edges that must be removed in order to isolate any sub-graph of i nodes. Based on this measure a graph on N nodes is made reliable by maximising $\delta(i)$ for all $i \in [1,N]$. Other measures based on the effect of edge or node removal on the diameter of a graph have been investigated by Wilkov [Wilkov 1].

The desirability of large girth for a given diameter and number of nodes in a regular graph was established by Wilkov [Wilkov 2]. Tutte [Tutte] has shown that the minimum number of nodes required to design a graph of given diameter and degree is a function of the degree and girth. This establishes an absolute minimum on the diameter of regular graphs with girth equal to $2k$ or $2k+1$ where k is the diameter. These are known as Singleton and Moore graphs respectively. It has been established that Singleton and Moore graphs have minimum diameter over regular graphs on n vertices of a given degree [Wilkov 1]. These results are very interesting since they define a lower bound on the diameters of regular graphs. However, these graphs only exist for a few combinations of degree and girth, and thus are limited to a few values of n (the number of vertices). The problem of finding the minimum diameter over all regular graphs of degree d has not yet been solved.

7.3.2. Circulant graphs

Maximum connectivity graphs are an important model for the design of reliable networks [Boesh],[Lipovski],[Wilkov]. Moreover, as mentioned above, many attempts have been made at finding the 'best' graph over all regular graphs. As far as parallel processing is concerned the ease of construction, the relatively low degree dictated by hardware concerns, etc. make the search for the ultimate graph less important than that required by the rigorous discipline of graph theory. In practice simplicity of construction together with a 'reasonable' diameter can make up for the distance away from the theoretical bounds.

A class of graphs known as Circulants [Harary] offers maximum connectivity properties for regular graphs. [Boesh] studied these graphs in order to design maximum connectivity, minimum diameter circulants. The latter have a very simple construction rule and are node symmetric.

The circulant graph on p vertices, $C_p(n_1,n_2,...,n_k)$ or $C_p(n_i)$ where $0 < n_1 < n_2 < \cdots < n_k < (p+1)/2$ has $i \pm n_1, i \pm n_2, \ldots, i \pm n_k \pmod p$ vertices adjacent to vertex i. The sequence (n_i) is called the jump sequence and the n_i's are called jumps.

Circulants have been studied and an optimality criterion based on the equality of the degree, edge connectivity and vertex connectivity verified for the case $C_p(1,....,k)$. One type of circulant is particularly interesting $C_p(n_1,n_2)$, because the vertex connectivity is four. Therefore, they can directly be used to construct multi-processors whose processing elements have four links e.g. the transputer. In fact the double ring network used in the ILLIAC-IV multiple processor system [Lipovski] is a $C_a2(1,a)$.

Because the diameters of circulants vary a great deal with the n_i's, Boesch and Wang have considered the problem of defining a lower bound. Since the graphs are node (or vertex) symmetric, without loss of generality they started from a vertex labelled 0 and constructed a tree in which each edge is labelled either n_i or $-n_i$ (there are 2k edges emanating from node 0). Moreover, they labelled the nodes reached through these edges n_i or $-n_i$. Obviously, this construction can lead to different paths reaching the same vertex. When that happens all but the first occurrence of a vertex are discarded. When all points at the same and different levels in the tree are distinct, the total number of nodes in the tree gives the maximum number of vertices that can be reached from node 0 using *m* edges (or jumps), where *m* is the depth of the tree. If X_m denotes the total number of nodes in the tree, to link the above assertion to the diameter of graph, the following condition must be fulfilled: $X_m \geq N > X_{m-1}$ where N is the total number of nodes in the circulant. If so the diameter of a circulant $C_p(n_1,n_2,...,n_k)$ is diag(C) $\geq$ *m*. Evaluation of *m* and X_m leads to the following theorem (theorem 2 in [Boesh]):

Let $G = C_p(n_1,n_2,...,n_k)$, if $X_m \geq N > X_{m-1}$ then diag(G) $\geq$ *m*, where

$$X_m(k) = 1 + \sum_{i=1}^{m} Y_i$$

$$Y_i = \sum_{j=1}^{\min(k,i)} C(k,j)C(i-1,j-1)2^j.$$

and C(x,y) is the number of combinations of y elements from an x element set.

The expression for X_m simplifies for k = 2 and k = 3 to:

$$X_m(2) = 2m(m+1) + 1$$

and

$$X_m(3) = 1 + (8m^3 + 12m^2 + 16m)/6.$$

The case k=2 was studied in [Boesh] and the bound attained for

$$m = \left\lceil \frac{(-1 + \sqrt{2N-1}}{2} \right\rceil$$

where $\lceil x \rceil$ is the smallest integer greater than x and N > 6. One of the major advantages of this formula, besides meeting the bound for this class of graphs, is that it provides the diameter explicitly. Moreover, these well behaved graphs exist for every integer value of the number of vertices in the graph N > 6.

Bevide *et al* analysed circulants with these characteristics and used graph isomorphism and a constructive method to map similar networks on a mesh topology with wrap-around links [Bevide]. The topology obtained shows, at least for cases where the network is a rectangle, that the wrap-around links connect nodes on opposite sides in a cross-diagonal manner Figure 7.5. This is, intuitively, the main cause for the good diameter properties over meshes with wrap-around links.

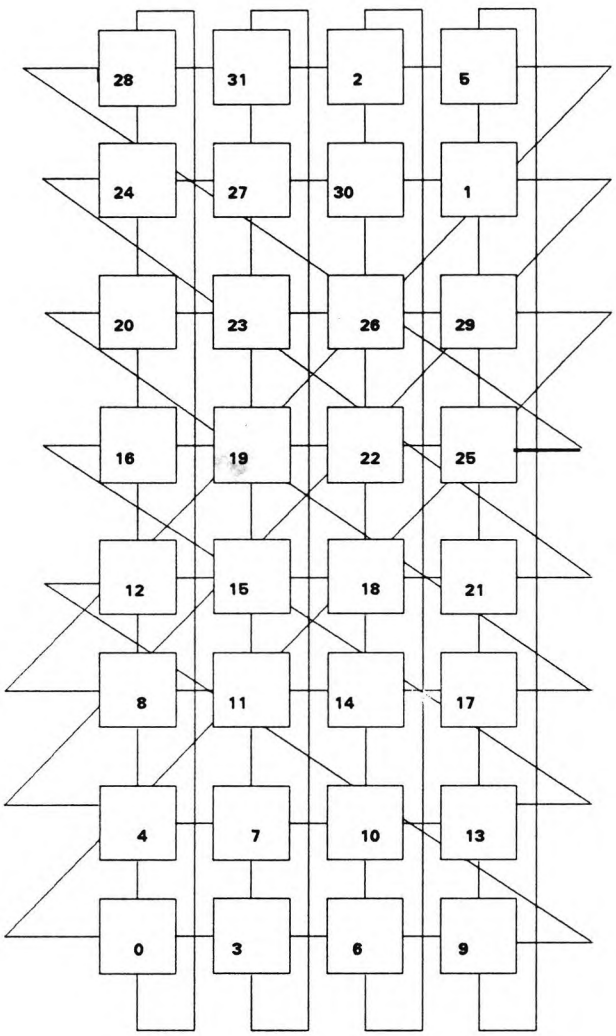

Figure 7.5: Mesh connected Circulant on 32 Nodes.

The proof for the isomorphism between $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ and the class of meshes derived can be found in [Bevide]. It is worth noting that all the optimality results obtained by Bevide *et al*, Boesch and Wang, etc. are only valid for the class of circulants. There are in fact, for some values of N, regular 4-graphs with smaller diameters, but the simplicity of construction makes these networks' implementation simple. Also generic functions can be simply defined for commercially available transputer and switch hardware and software. Viewing $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ as a mesh makes the implementation of image processing routines simple. Besides, the

small diameter of the circulants and the simplicity of optimal routing should be an advantage for the higher level vision tasks that require global data.

For the design of relatively small networks with a given number of nodes, a procedure which builds a rectangular mesh and then uses permutations of the links available on the boundary to minimise the diameter of the network was defined. It uses an algorithm due to Floyd [Floyd] to evaluate the minimum distance between all pairs of vertices in a graph. The largest such distance is the diameter of the graph. Floyd's algorithm together with the choice of permutation groups adopted and aspects of the geometry of the mesh are given in appendix D.

7.3.3. Architecture for large multi-processor systems

De Bruijn graphs DB(2,N) have excellent diameter properties, however, they are not regular and cannot emulate meshes. In the SIMD model most of the algorithms rely on regular structures that are implemented on lattices, and hence are realisable on a mesh. Another important aspect in the MIMD model is the fact that even small distances in a store and forward communication network can be taxing. Therefore, changing the inter-connection network dynamically can be beneficial. However, implementing dynamic switching introduces further overheads and eventually a bottleneck if the switching resource is centralised. A concept that helps in clarifying the situation is that of **inductive architectures** [Lipovski]. Lipovski and Malek define the hardware (processors and switches) in a multi-processor system as input energy and the amount of computational power as output energy. Then, an architecture is inductive if the output energy is a strictly increasing function of the input energy, as N (the number of hardware modules) is increased from N to N+1 as $N \rightarrow \infty$. Obviously, induction is a desirable property. It is defined similarly for algorithms. Therefore, algorithms should seek induction on inductive architectures.

For a dynamically switched network, the management of the switches can be considered as non-processing hardware (energy). Therefore, if the complexity of the switching mechanism is proportional to the size of the network N, the architecture cannot be inductive.

To simplify the problem the following method is put forward. First, a basic architecture exhibiting good characteristics in terms of the measures introduced above is defined. Then a switching network is introduced to operate on a subset of edges when nothing but a direct link would do. Of course, defining such an architecture is not enough to

guarantee optimal performance, but it would be a good testbed for analysing parallel algorithms and distributed switching and scheduling procedures.

An architecture fulfilling the conditions stated above is now introduced. It is based on the optimal circulant $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ connecting nodes that are De Bruijn graphs DB(2,3). The rationale behind this design is that both $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ and DB(2,3) have simple building rules and good diameter and average distance properties. Moreover, DB(2,3) or an undirected version (Figure 7.6) has six free links. Four of the latter will implement the circulant and the two others can be switched to allow rapid access to a particular region of the multi-processor system. In effect, such a network builds on the sound graph theoretical characteristics of two models to produce a two-level structure. From the communication point of view, data transmission is seen as a two-stage process; namely, a message is transferred along the circulant links until it reaches the destination group where a second mode is used to reach the destination processor.

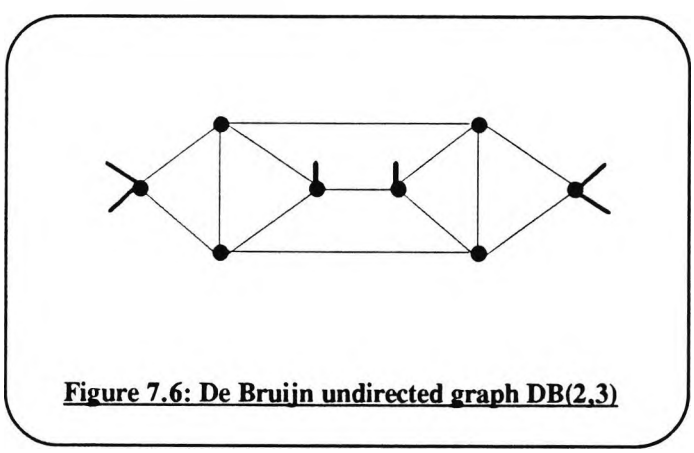

Figure 7.6: De Bruijn undirected graph DB(2,3)

The composite node DB(2,3) is ideally suited as vertex in an optimal circulant, due to its small local diameter and relatively low connectivity. An optimal routing algorithm can be designed for $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ by considering the distance between two nodes as a function of two variables on a two-dimensional finite lattice, and using a maximum gradient descent criterion to reach the minimum 0 (when a message reaches its destination).[8]

The composite nodes of this architecture are not node or edge symmetric, therefore, the choice of free links to implement the $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ is critical. The routing algorithm mentioned above is instrumental in effecting this choice. The distance

[8]See chapter 8 for the details, algorithm and Occam implementation

between two nodes is travelled first through the connections corresponding to $\lceil \sqrt{N}/2 \rceil$ and then $\lceil \sqrt{N}/2 - 1 \rceil$. Any shortest path between two nodes contains only one switch in direction. Therefore, keeping the good diameter and average distance properties of $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$ suggests using the two nodes with two free links each (Figure 7.6) to implement the circulant connections. The effect of this set-up is a bounded increment to the diameter of the simple circulant (on single processor nodes) while increasing the number of nodes in the network eight-fold. Admittedly, the network is not optimal in the sense that the node and edge connectivities are no longer maximum. However, the two free links of the central nodes can be switched into a global store or provide fast distributed access to the world beyond the computer system. The two free links of the central nodes were chosen as switching connections for two reasons:

1. The average distance to the other nodes is smaller than that of the end nodes.
2. The other free connections are ideally suited for the circulant connections because data is delayed by entering the DB(2,3) only when it is changing circulant direction or it is destined to a local node.

It is worth noting at this juncture that the network proposed here outperforms $C_N(\lceil \sqrt{N}/2 - 1 \rceil, \lceil \sqrt{N}/2 \rceil)$. Table 7.1 presents the comparative diameters properties of several topologies.

The topology proposed is a De Bruijn graph based circulant or DBC(N) where N is the number of nodes in the graph. N must, therefore, be of the form N = 8p where p is an integer. This is a drawback in graph theoretical terms since it does not exist for every value of N. However, as far as multi-processor computer systems are concerned, the ever decreasing cost of microprocessors warrants the use of large numbers of processing elements. Besides, the composite node is standard and can be built for transputer systems as a TRAnsputer Module TRAM [Inmos 1].

Nodes	8	64	512	1024	8192
Hypercube	3	6	9	10	13
De Bruijn	3	6	9	10	13
Circulant	2	6	16	23	45
Torus	3	8	24	32	96
2D mesh	4	16	32	64	512
Proposed architecture	3	10	14	16	31

Table 7.1: Diameters for different number of nodes

It is believed that a successful architecture for computer vision should provide facilities for both the SIMD and MIMD implementation of algorithms[9], since, often computer vision tasks can benefit from both models of parallelism. Such an architecture presents a difficult problem because the SIMD and the MIMD models rely on entirely different concepts. For example, synchronisation at the instruction (or task) level is a means to achieve high performance in the SIMD model, but it is a nightmare in the MIMD model.

Besides, the architecture should have a high input/output (IO) bandwidth. This not only increases the rate at which input data and results are written to or read from the system, but also allows easy integration of application specific sub-systems. For computer vision, these are framestores and mass storage devices. Another very important aspect is the interpretation of the notion of diameter (above). Very often a particular processor is more "important" than the others (e.g. the processor coupled with a frame store). In this case, and supposing an SIMD type algorithm, the diameter of the network is no longer a valid measure of communication latency, instead the maximum and average distances from this processor to the rest of the network becomes more relevant.

7.3.4. Dynamic Reconfiguration

Multi-processors based on the message passing paradigm use either the store and forward or the wormhole (or cut through) routing techniques [Garcia].

The store and forward technique relies on buffering facilities provided by intermediate nodes in the path of the message. This introduces message latencies that are proportional to the distance travelled by the message. Besides, task distribution, and theoretical methods ([Robinson]) become impractical as analysis tools for performance optimisation.

The cut through routing technique relies on special hardware to start forwarding the message as soon as a header containing the identity of the addressee is read. In this mode the effects of the diameter on latency are reduced, since the delay introduced by passing through intermediate nodes is very small. However, the message is no longer a unit since it can spread over several processors. Lau and Lau [Lau] demonstrated that,

---

[9]A transputer system, in fact, supports the SISD model, in other words, the performance of a single transputer is comparable to the best micro-processors of its generation.

although wormhole routing is not supported by the Transputer (T800), it can be implemented (in Occam). However, as noted in [Peel], the implementation does not comply with the basic rules of Occam since two processes access a single buffer simultaneously.

Many authors have stressed the importance of reconfiguration of the interconnection network for MIMD computer system [Garcia][Duato][Jones][Nicole]. Garcia and Duato advocate the use of limited reconfiguration by keeping the 'backbone' of the topology and switching a subset of links. This approach is well suited to the network described above. However, their algorithm relies on a control bus (and so does that of [Jones]) that allows individual processors to require reconfiguration from a controller node. The controller in turn informs all the nodes in the network that a change is imminent (also through the bus), performs the necessary changes, and then broadcasts an 'OK to proceed' message. The net effect is that a scheduling point is introduced and the excellent results for numerical analysis examples, will only be obtained for cases where the cost function [Garcia] is representative of the communication patterns as well as the amount of traffic.

DBC(8p), where p is an integer, can be viewed as a two level system: a static structure which consists of a circulant on Composite Processing Elements (CPE) and a fully reconfigurable 2-connected network on CPEs.

7.3.4.1. Distributed reconfiguration scheme

As the number of processors in a parallel system increases, attention should be directed towards the overall reliability of the system. After a while a processor (or a few processors) might fail. A relevant question is what is the effect on the performance? Therefore, a simple mechanism for diagnostics and fault reports should be included in the design. In this section, a scheme is proposed for the dynamic reconfiguration of the network proposed in section 7.3. The hardware involves the extension of the network with C004[10] crossbar switches. Without loss of generality the reconfigurable network described next is assumed to consist of 512 transputers.

First, let us define the basic switching requirements. DBC(8p), where $p = 64$, consists of a circulant of 64 CPEs. The circulant connections form the static configuration and

[10]see appendix A for the details of the Inmos C004 crossbar switch.

will not be further considered in this section. Each CPE has two free links that are used in the configuration set-up. These two links are considered functionally as:

- a request and diagnostic link and
- reconfiguration link.

Due to the symmetry of the CPEs and the central position occupied by these two links, no further distinction is required.

Second, the desirable features of the scheme are as follows:

- Ease of implementation,
- adherence to the point to point communications model,
- standard building blocks
- efficient diagnostics capability,
- reduced delay

The first three points are addressed by the design of the Inmos C004. The inputs and outputs of the latter implement the same link protocol as the transputer. Therefore, no extra logic is required to connect the switches to the network. Besides, each device consists of 32 to 1 multiplexors and once a route is established between two links the connection is completely independent. Finally, the C004 is commercially available.

The fourth point involves the ease of access to individual processors in the network for diagnostic purposes. To this effect, and in order to provide a route for the reconfiguration requests, the switching system is divided into two identical subsystems. Hence, the two free links are considered functionally as a control link and a data link. Figure 7.7 shows a switching subsystem. For the system considered here (512 nodes) each subsystem is made up of 12 C004s connected in a shuffle network .

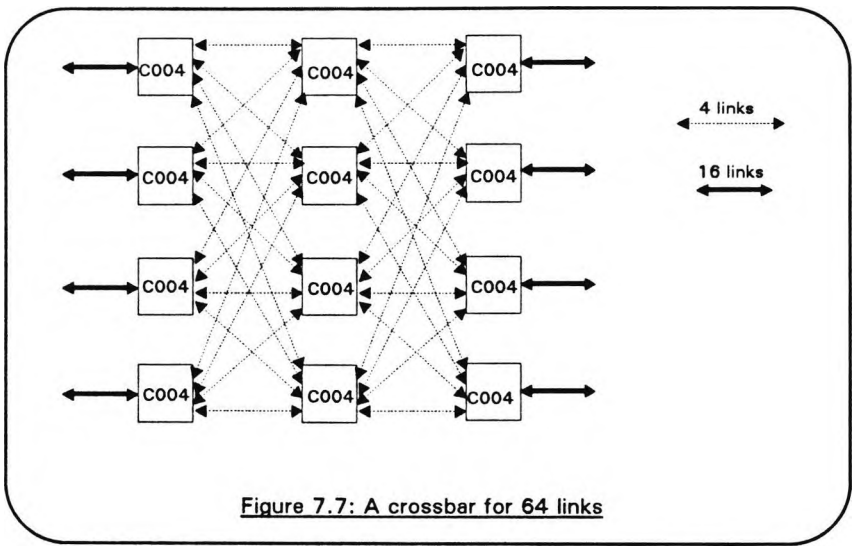

Figure 7.7: A crossbar for 64 links

This crossbar provides full reconfiguration on 64 links. Therefore, two such crossbars are needed for the scheme described here. The first subsystem provides connections between a link on the controller processor and the configuration links of the CPEs, and the second implements the reconfiguration. Code on a transputer requiring a connection to another part of the network sends its request to the transputer which contains the request link and it sends the message to a buffer on the transputer which contains the reconfiguration link. Figure 7.8 shows a CPE with identified reconfiguration links.

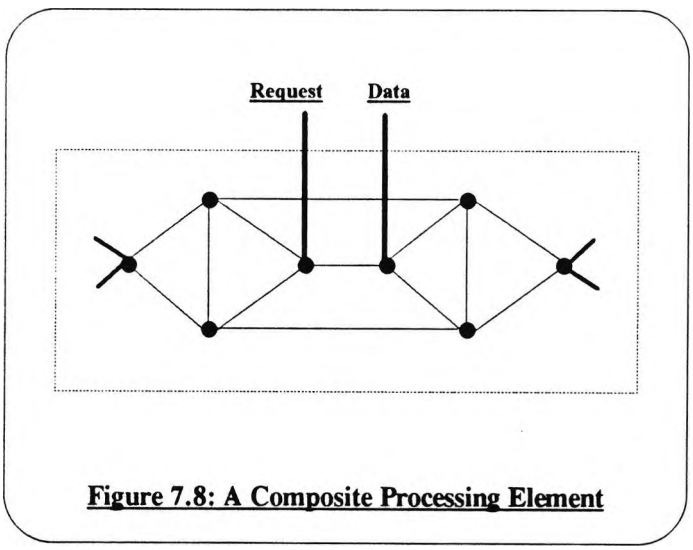

Figure 7.8: A Composite Processing Element

Figure 7.9 is a block diagram of a system where only the two links on the CPEs are shown , each switch corresponds to the configuration described in figure 7.7.

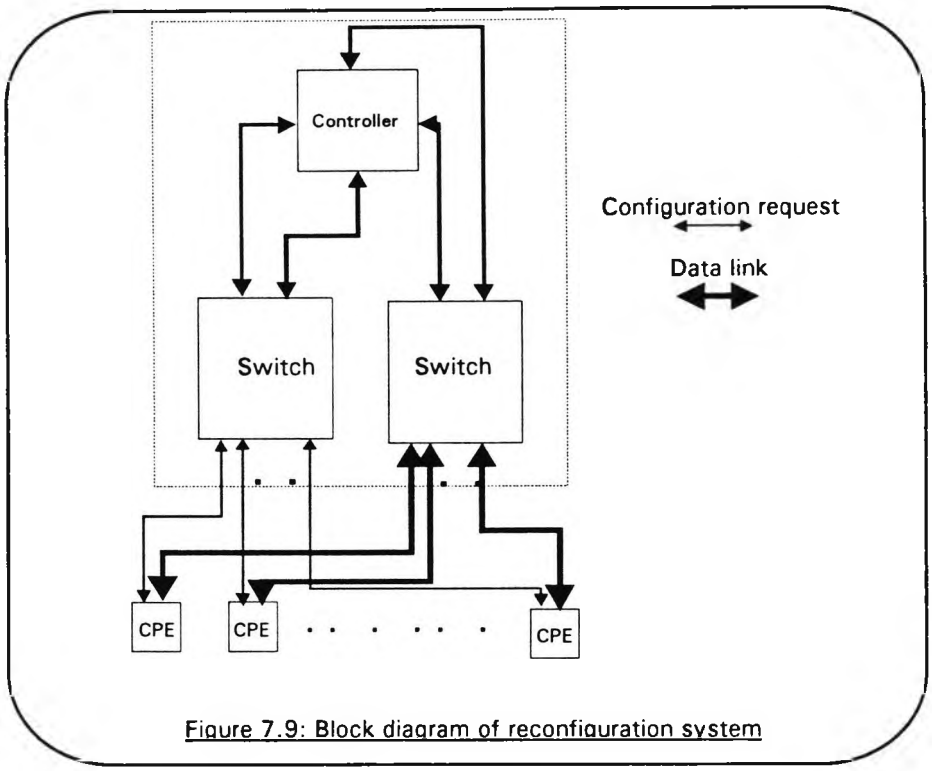

Figure 7.9: Block diagram of reconfiguration system

The switch subsystem which handles requests can do so under the control of a transputer which operates on all switches. The controller handles requests in a round robin fashion. This introduces delay in granting requests to transputers in the network but it must be seen in the context of a parallel system where requests are relatively infrequent. In other words, reconfiguration is not needed at the instruction level. Besides, the C004 implements a simple link protocol (appendix A) which can be used to effect reconfiguration of a link in a few instructions. Crossbars[11] of arbitrarily large sizes have been reported [Inmos 1]. However, the configuration proposed here differs in the fact that reconfiguration relies on the synchronisation of requests. In other words, there needs to be a mechanism which ensures that requests are not lost (or worse, cause deadlock) if the controller does not respond within a specified time. This seems to defeat the purpose of the decentralised point to point reconfiguration. The problem is akin to that of communication through an unreliable channel [Inmos 1]. Therefore, an adaptive method can be devised whereby a switching request is made according to information gathered from different sources (token specifying the last time the switch controller polled, time for the expected next poll, cost of using circulant connections, etc.)

[11]Using C004s

The basic advantage of such a decentralised switching system is that no global synchronisation is necessary. Although, a waiting time is experienced when requests are made or connections are relinquished, this only affects the two processors engaging in data transfer.

7.4. Summary

This chapter introduced a network topology suitable for a transputer based computer vision system. The main features of this architecture are:

- a relatively small diameter and average distance,
- a regular structure,
- a reconfiguration sub-network.

The first point above is a desirable property in all parallel computers since the diameter and average distance affect the overall communication delay. The second point benefits low-level vision algorithms which involve localised communications. Finally, the third point is beneficial to the intermediate to high-level vision algorithms which are likely to require global and time-varying communication patterns.

The simple dynamic reconfiguration scheme presented allows for processors in the network to request reconfiguration individually without the need for global synchronisation. The switching sub-network can be extended to include access to a main store containing data which can then be considered distributed[12].

The advent of router chips (e.g. C104) introduces a new approach to the design of communication networks. The universal message passing parallel computer [May], establishes full connectivity between all processors and therefore, does not rely on any given topology. The advantages of this approach are numerous [May]. However, for a large network the number of routing chip between two processors can become large, thus introducing more delay in communication paths. It is believed that the topology presented in this work, running code developed specifically for it, can outperform the general purpose universal machine. Also, the router can be used in lieu of the crossbar switch to improve the dynamic reconfiguration scheme.

---

[12]This is only true if the delay introduced by the switching mechanism is bounded and significantly smaller than the data transfer time. This is very likely to be the case for large networks.

CHAPTER VIII

8. Routing algorithm, performance evaluation and practical issues

This chapter introduces an optimal routing algorithm for the architecture introduced in section 7.3. This algorithm relies on the fact that architecture can be seen at two levels (inter and 'intra' CPE). The inter-CPE communication is carried out along shortest paths due to the monotonicity (a characteristic of circulant graphs) of the running distance between the destination node and the current node. The intra-CPE communication takes advantage of the simple building rule of De Bruijn graphs. The basic idea is to use the circulant connections (of CPEs) judiciously in order to traverse at most one CPE when forwarding a message between two nodes.

Also a performance monitoring program is described together with some Occam implementation issues. This monitor can be run in parallel with application code to evaluate the efficiency of processors..

8.1. Routing algorithm for DBC

First, the routing algorithm is deduced from the structure of the circulant and De Bruijn connections of the proposed topology (DBC). Then, the Occam implementation is described. The latter consists of a communication manager which runs in parallel with application code on every processor in the network. Figure 8.1 depicts the internal structure of a transputer in the network. A routing process handles all communication issues and a worker process performs application computations. The main purpose of the buffer process is to avoid blocking in the router while waiting for

the worker to consume data. In fact, the routing process handles both the circulant connections and communication between the processors of the same CPE. Therefore, the routing algorithm has to make the following decisions:

- Has the packet reached its destination ?
- If not, is the communication to proceed along the same direction of the circulant ?
- Is the communication to change direction ?
- Is the communication amongst the processors of the same CPE ?

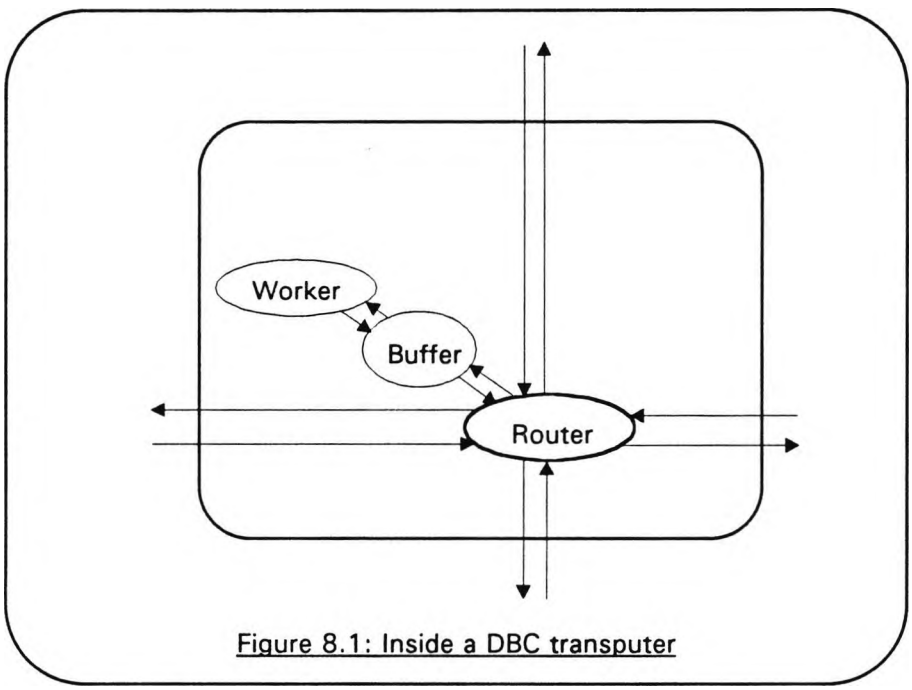

Figure 8.1: Inside a DBC transputer

Therefore, the routing task consists of two different sub-tasks: one to deal with the circulant connections and the other to handle CPE connections. In the next few sections first the circulant connections are dealt with then the CPE connections are considered and finally the two are brought together in an algorithm that performs routing along the shortest routes in a DBC network.

8.1.1. Routing along the circulant connections

Since the circulant graph is vertex and edge symmetric and given the node numbering introduced in chapter 7, a circulant can be seen for distance measurement as an infinite 2-dimensional lattice. The latter is characterised by the jump lengths m and $m+1$ and the infinity analogy is derived from the fact that only distances between pairs of nodes (arbitrary) are required. Figure 8.2 shows such a lattice and identifies a shortest route between two nodes. Because the lattice is 2-dimensional the function describing the

distance between two nodes must be monotonic. Therefore, a maximum gradient descent approach will yield a shortest route between any given pair of nodes. This distance is given by the number of jumps required to move a datum from a source node to a sink node. It must, therefore, take the form: d=a*m+b(m+1) where m is the characteristic number of the circulant and, a and b are the number of jumps of size m and m+1 respectively. Also, the distance is the difference between the identifiers of the destination processor and the source (or current) processor. Because jumps of size m+1 reduce the distance by a larger amount they are considered to yield the maximum gradient. Therefore, the routing process should convey messages first along the connections of the m+1 jumps then along the jumps of size m.

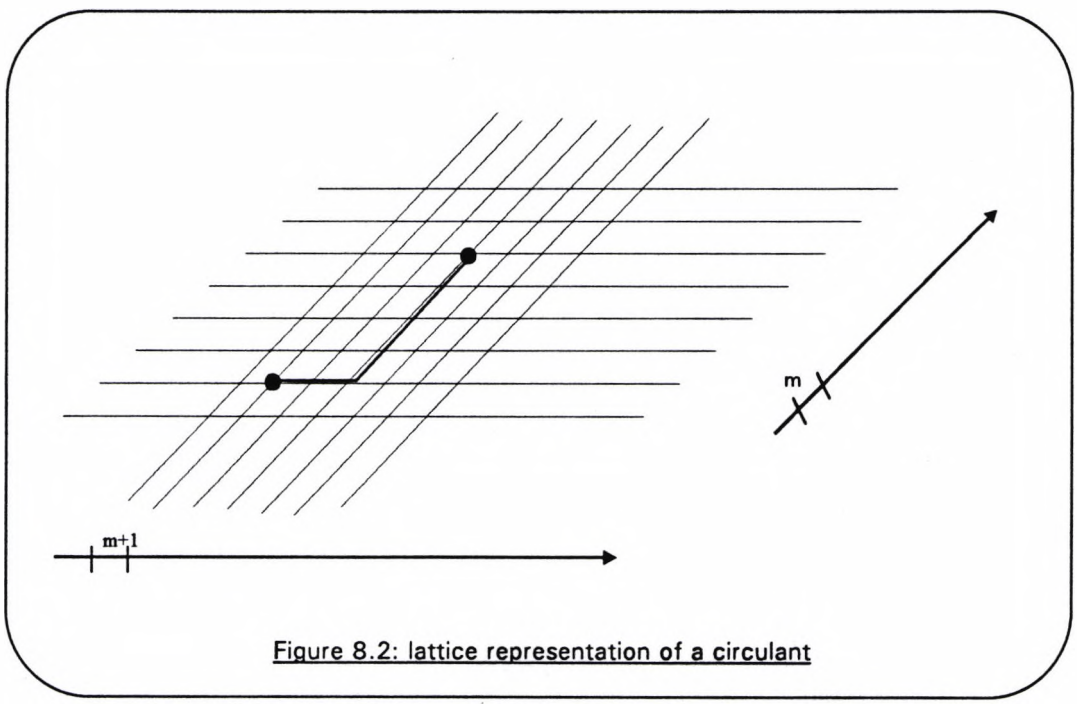

Figure 8.2: lattice representation of a circulant

This ensures that routing proceeds along the direction of maximum gradient. The change of direction should happen when the difference between the address of the destination and that of the current node are a multiple of m apart[1]. At this point the distance between the destination and the current node can be covered exclusively with jumps of size m.

Therefore, a simple routing algorithm can be defined by performing the following operations at each node in the network:

[1]To resolve the case where the distance is a multiple of both m and m+1, the jump of size m+1 takes precedence because of the larger gradient.

- Get message (either from the local worker or from one of the links),
- decode the destination address and find the distance $d = d_d - d_c$ (where d_d is the destination node identifier and d_c is the current node identifier),
- If $(d > N/2$ or $0 > d > -N/2)$ then set direction *dir* = -1 else *dir* = +1
- If $|d| = 0 \bmod (m)$ then
  - If $|d| \neq 0 \bmod (m+1)$ then *jump* = *m* else *jump* = *m*+1
- else *jump* = *m*+1
- If d = 0 destination reached else forward message along jump in direction *dir*

Note that the setting of the jump identity follows from the maximum gradient criterion. The direction, on the other hand, is derived from the symmetry of the circulant.

Before presenting an Occam implementation of this routing algorithm let us associate the two jump sizes with the transputer links. Figure 8.3 shows a transputer with links identified with jumps in the circulant configuration.

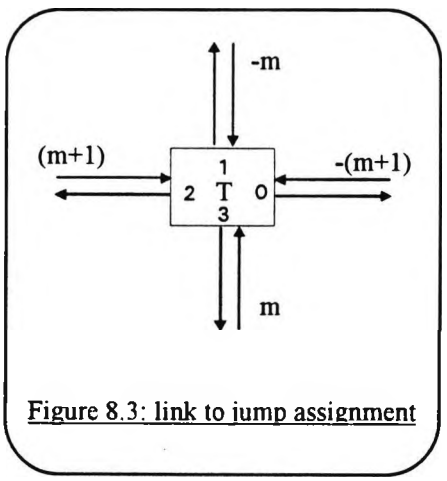

Figure 8.3: link to jump assignment

The following listing presents an Occam implementation of the routing algorithm described. The messages are supposed to have a fixed length[2] (a header specifying the addressee's number). An additional channel pair (besides the four links) is declared for communication with the buffer (see figure 8.1).

Listing 8.1

```
VAL INT N IS 512:
VAL INT Nby2 IS (N/2):
```

[2]Note that variable length messages can be handled by including the message length in the header.

```
VAL INT m IS 16:
VAL INT mP1 m+1:
VAL INT length IS 1024:
PROC router( VAL INT source, [5] CHAN OF ANY in, [5] CHAN OF ANY out)
[length]INT buffer:
INT dir,dest:
WHILE TRUE
  ALT i=0 FOR 5
    in[i] ? dest; buffer
      SEQ
        dist := dest - source
        IF
          dist <> 0  -- The message has no reached its destination
            SEQ
              IF
                ((dist > Nby2) OR ((dist>-Nby2) AND (dist < 0)))
                  dir := -1
                TRUE
                  dir := 1
              IF
                (((dist REM m) <>  0) OR (((dist REM m = 0)) AND ((dist REM mP1 = 0))))
                  IF
                    (dir <0)
                      out[0] ! dest; buffer
                    TRUE
                      out[2] ! dest; buffer
                TRUE
                  IF
                    (dir <0)
                      out[1] ! dest; buffer
                    TRUE
                      out[3] ! dest; buffer
          TRUE
            out[4] ! buffer
:
```

Were the network a circulant the above listed router would have conveyed messages between nodes along shortest routes. The following code fragment shows an example of the code on each transputer of such a network.

Listing 8.2

```
PRI PAR
  router(id, chanin, chanout)
  PAR
    store(chanout[4], chanin[4], from.worker, to.worker)
    worker(to.worker, from.worker)
:
```

The transputer can execute processes at one of two priorities (see appendix A for details). In Occam the PRI PAR construct takes two component processes and executes the first one at high priority and the second at low priority. In the previous listing the router is executed at high priority. This is a popular technique amongst transputer users, which ensures that communication requests are dealt with as soon as possible to avoid long waits for data.

8.1.2. Routing within a CPE

CPEs consist of 8 transputers connected in a De Bruijn DB(2,3) graph configuration. The simple construction of the De Bruijn graphs suggests yet another simple algorithm for routing messages. The latter is based on the binary representation of the numbering sequence which identifies each processor.

The construction of DB(2,3) proceeds as follows. The nodes are labelled with 3-bit binary numbers[3]. A directed edge is present between node a $(a_2a_1a_0)$ and node b $(b_2b_1b_0)$ if the two most significant bits of b (b_2b_1) are equal to the two least significant bits of a (a_1a_0). Therefore, the nodes of the CPE shown in figure 7.8 can be labelled as follows (Figure 8.4 where directions, parallel edges and loops are discarded).

[3]In the general case DB(n,D) nodes are labelled with words of length D on an n-letter alphabet.

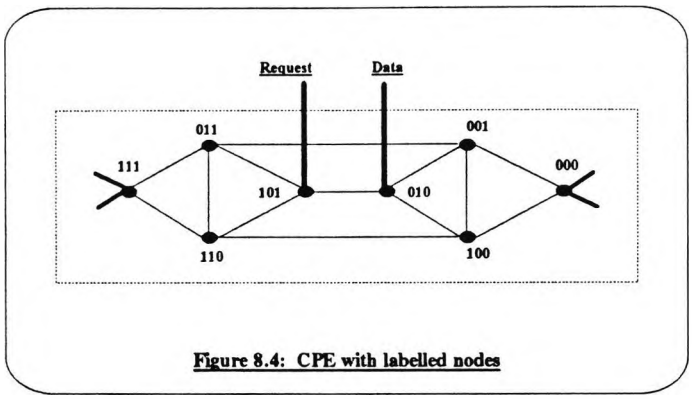

Figure 8.4: CPE with labelled nodes

Next, a conjecture[4] defines a better routing algorithm for De Bruijn graphs. This algorithm sends messages between pairs of nodes along shortest paths. The method relies on the direction of edges in (the directed version of) DB(2,3) (Figure 8.5).

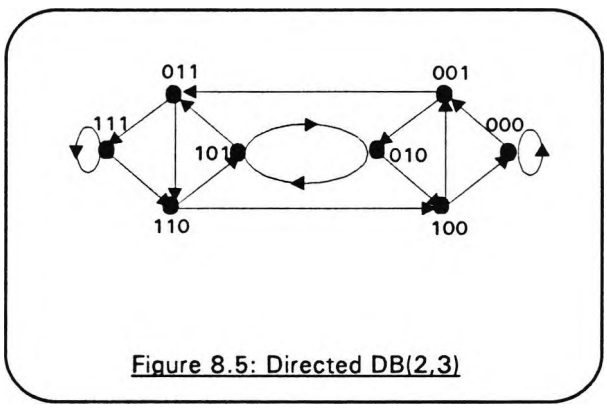

Figure 8.5: Directed DB(2,3)

Before stating the conjecture let us consider a simple routing algorithm for DB. The latter [Liu] dispatches messages using 'left shift' operation. In other words, a message travelling from node $a_2a_1a_0$ to node $b_2b_1b_0$ follows the path[5] ($a_2a_1a_0$, $a_1a_0b_2$, $a_0b_2b_1$, $b_2b_1b_0$). This algorithm is suggested by the construction of De Bruijn graphs which, therefore, guarantees the existence of the path. However, there is no guarantee that the path thus described is the shortest possible in the undirected version of the graph. For example, consider the path between node 3 (011) and node 1 (001) defined by the above algorithm (011, 110, 100, 001). By ignoring edge direction (as in CPE) this path can be changed to (011, 001).

---

[4]The conjecture is given for a general DB(2,D.) As far as DB(2,3) is concerned this can be verified.

[5]This is the longest path generated by this algorithm since at most three left shift operations are required to transform the label of a node into that of any other node in the graph. As an example of a shorter path consider the path (000, 001, 010) between node 0 (000) and node 2 (010.)

Conjecture:

An optimal routing algorithm for DB(2,D) can be stated as follows:

Define an alternative shift operation as follows: a message travelling from node $a_2a_1a_0$ to node $b_2b_1b_0$ follows the path $a_2a_1a_0$, $a_1a_0b_0$, $a_0b_0b_1$... Then select the shift operation to apply based on the shortest cycle covered by the two operations. For the undirected version of the graph another decision is made as to whether the cycle is to be covered in one direction or the other. Note that this introduces a pair of right shift operations. The existence of the alternative cycles introduced follows from the construction of De Bruijn graphs.

As an example of the way this works consider a message sent from node 4 (100) to node 1 (001). With the original algorithm the message will be despatched towards node 0 (000) then around the self loop of node 0 and finally to node 1. With the second shift operation the message will be sent straight to node 1. However, even if the conjecture were true it would not provide an algorithmic solution to the routing problem. Nonetheless, for a given De Bruijn graph the cycle calculations can be done statically and included explicitly in the routing algorithm to effect choice.

Following is a listing of an Occam program implementing optimal routing in node 0 of DB(2,3), identifying the two links connecting node 0 to node 1 and node 4 as up and down respectively.

Listing 8.3

```
CHAN OF ANY local.in, local.out, up.in, up.out, down.in, down.out:
PROC router (VAL BYTE id)
VAL []BYTE route IS ['u', 'u','u','d','d','d','d']:
SEQ
  ALT
     up.in ? dest; message
     down.in ? dest; message
     local.in ? dest; message
  IF
    (dest <> id)
        IF
        (route[dest] = 'u')
            up.out ! dest;message
        TRUE
```

```
            down.out ! dest; message
    TRUE
            local.out ! message
:
```

In the above listing, only the route array is changed from one node to the other.

8.1.3. Combined Router for DBC

Routing messages in the DBC follows directly from the algorithms introduced in the previous two sections. First, node 0 (000) and node 7 (111) of each CPE are the only two processors that handle the circulant graph routing task. Figure 8.6 shows a CPE with the circulant connections identified.

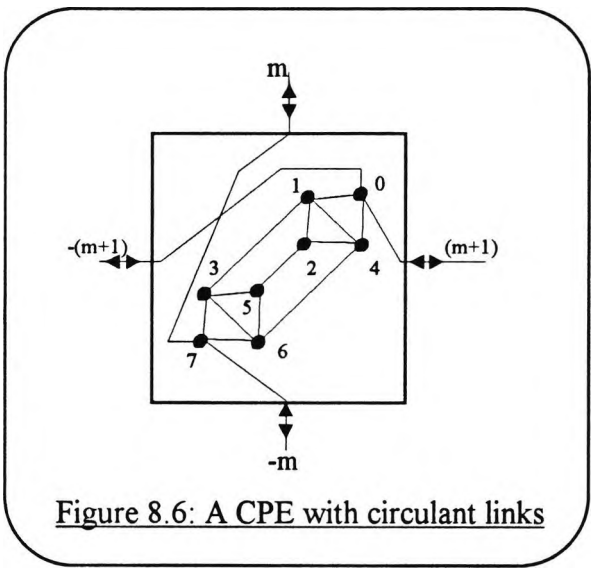

Figure 8.6: A CPE with circulant links

In the DBC, nodes 0 and 7 of each CPE implement the routing algorithm of listing 8.2. Nodes 1-6 implement the algorithm depicted in listing 8.3. Some additional code is required to interface between the two modes of routing. Since all individual processors in the network need to be aware of the configuration a common protocol needs to be established. The address of a node should comprise two fields one for the CPE address and one for the destination processor within the CPE. Listing 8.4 shows an example of such a protocol.

Listing 8.4

```
PROTOCOL message IS INT16; BYTE; INT; []BYTE:
```

In this protocol the first entry is a 16-bit integer used to hold the CPE address, the second entry specifies the processor address within the CPE, the third entry indicates the length of data and the last is an array containing data.

A DBC running the routers developed in this section transfers messages between any two transputers along the shortest route. A message issued inside a particular CPE will be forwarded to the appropriate port (node 0 or node 7 of the same CPE) according to the circulant hops. Then, the circulant routing is initiated and carries the message to the destination CPE.[6] Once at the destination CPE, the routing switches back to the shift operations to convey the message to its final destination.

8.2. Performance monitoring

Jones *et al* [Jones 1] have proposed an algorithm (supervisor) for the measurement of the percentage of time the transputer is idle during the execution of an application program. The method depends on the way in which the scheduler functions (Appendix A). The algorithm defines two processes that are run in parallel. A CPU[7] load counter process (CLC) and an idle loop counter process (ILC). First, a calibration stage is run before the application program is allowed to start. This is to ascertain that no other process interferes with the calibration. At this stage, the number of times the main loop of ILC is run in a given period (P) is evaluated. This number is the totally idle count (IC_0) i.e. the value of the count for a processor that is completely idle. Second, the supervisor is started and the application program is allowed to run. The start command is received by CLC which, in turn, notes the time by reading the current value of the timer and sends a start command to ILC. The latter clears its counter and proceeds to measure the number of times the processor is idle during the execution of the application code. This is done by forcing the rescheduling of ILC thus giving a chance

[6]In cases where both jumps are used a CPE along the path will have been crossed. This is an operation equivalent to sending a message within a single CPE with the source and destination nodes being 7 and 0 an vice versa.

[7]Central Processing Unit.

to every other process to be scheduled. If ILC is rescheduled no later than the time it takes the scheduler to perform task switching (a threshold value dependent on the processor speed, etc.), IC is incremented. When the application code terminates the values of P, IC_0 and IC together with the elapsed time are used to evaluate the percentage of time the processor was found to be idle during the execution of the application code.

One disadvantage of the above method is its dependency on a threshold determined by the processor type and speed. Consequently, accurate data about the processor, clock cycle, etc. must be available.

In order to avoid this problem and to simplify the design of the performance monitor the following method is adopted. Consider a process that can only run if no other process is ready. If this process is capable of measuring the amount of time T_R it spends running, then, running this process in parallel with some application code and comparing the total time elapsed (on completion) to T_R will provide a measure of the idleness of the processor.

The monitor consists of a single low priority process which runs continuously as long as the low priority process queue is empty and no high priority process is ready. As soon as a low priority process becomes ready the monitor deschedules and puts itself at the back of the ready queue. The code fragment in listing 8.5 demonstrates the idea. The descheduling operation is implemented through the timer input time1 ? now PLUS 15. The number 15 of low priority timer ticks ensures that the process is rescheduled before the minimum time for a time slice (1 millisecond) has elapsed.

Listing 8.5

```
WHILE needed
  INT val, dummy:
  TIMER timer0, timer1:
  SEQ
    timer0 ? start
    GUY                    -- GUY construct (to allow machine code) to store the
      LDLP val             -- pointer to the first process in the low priority queue
      SAVEL          -- in the variable val
    IF
```

```
(val = NotProcess.p)          -- If this pointer is NIL goto the timer0 ? end instruction
  SKIP                -- to calculate the time
TRUE
  SEQ
    timer1 ? now   -- Deschedule  this process
    timer1 ? now AFTER 15
    timer0 ? start
timer0 ? end
totalticks := totalticks + (end MINUS start)
```

Note that this process must run at low priority since a high priority process will interrupt the other tasks.[8]

Since high priority processes do not conform to the description which forms the basis of the algorithm described, further consideration must be given to the case when the situation arises. However, since a process is only interrupted at specified instructions the state of the program between the two timer0 inputs is quite stable.

8.3. Summary

This chapter presented a routing algorithm for the architecture introduced in section 7.3. The algorithm is optimal in that it conveys messages along shortest routes in the network. Only a simulation on one transputer was tried since a 'real' run requires a large number of processors. Each transputer was implemented as an Occam process consisting of the routing process a packet consumer and a packet producer. A control process communicating with every other process was also implemented. The latter is the interface between the system and the user who can instruct any process to issue a message (to any destination). The message is empty at first and consists of the source and destination processes. As it is forwarded from one process to the next the intervening processes append their identification numbers. When the message reaches its destination the control process reads it and displays the route which can be checked against a drawing of the network.

---

[8]The transputer scheduler operates on low priority processes only. High priority processes always run when they are ready and are never time-sliced. Therefore, performance monitoring which should be as little disruptive as possible is run at low priority.

The purpose of the simulation described above was to ascertain that messages are dispatched along minimum length paths using the proposed routing algorithm. For more thorough simulations a simulator such as Transim [Hart] could have been used. This was not carried out because the simple simulation is enough to arrive at the result pursued. Transim is a parallel design tool which simulates an application program as it runs on a network of Inmos transputers, the input being a subset of Occam with special timing commands built-in. Parallel execution, alternation, channel communication, timeslicing, priorities, interrupts, concurrent operation of transputer links, effects of external memory and so on are taken into account.

Similarly to the topology developed in this work, routing schemes for specific applications can benefit from the latest software developments such as TINY (University of Edinburgh) or the Virtual Channel Router (VCR) [Debbage]. VCR is a software package developed at the University of Southampton to provide unrestricted channel communication across networks of T4 and T8 series transputers. It allows distributed transputer programs to be written, compiled and configured in a topology-independent format and then bound to a topology-dependent routing kernel at run-time.

Besides the routing algorithm, a performance monitor is also presented in this chapter. It was used extensively and the results were found to be comparable to those of the algorithm given in [Jones 1]. However, the formulation of the proposed algorithm is simpler and does not require calibration or processor dependent parameters.

CHAPTER IX

7. Conclusions and discussion

The work presented in this thesis was primarily concerned with the design of a transputer network topology which is compatible with the wide range of requirements in computer vision. First, the inadequacy of the transputer for low-level image processing is noted. This inadequacy is linked to the processing power to communication bandwidth ratio. The latter was seen, in the literature, to give a rough estimate of the range of 'grain' that a particular processor can handle. The advent of very fast 1-D and 2-D multipliers and convolvers[1] made some low-level image processing tasks at frame rate possible. Therefore, a medium-grain processor like the transputer seems ill-suited for the whole spectrum of applications in computer vision. These fast multipliers can be considered as very fine-grain multi-processors since they derive their high performance from the fact that many similar and simple operations are executed simultaneously on separate chunks of data and synchronisation takes place at the instruction level. Thus, they implement data parallelism religiously and achieve impressive results. Unfortunately, this kind of model does not fit other levels of the hierarchy of computer vision tasks. Few algorithms present 'virtually no' sequential part[2]. Therefore, a system well suited for computer vision must present features facilitating both the data parallelism and functional parallelism approaches.

Having established the advantages and shortcomings of the transputer, attention was turned to designing communication networks that present a good compromise with regard to some measures of communication latency and delay. The latter are seen to depend on the communication bandwidth of the processor, the number of

[1]For example the Inmos Axxx series

[2]The convolution of an image with a 3x3 kernel for enhancement or edge detection purposes contains very little sequential operation (the multiplication of a pixel with a kernel coefficient); besides, it requires only local communication since the output pixel only depends on the input pixel and its immediate neighbourhood.

communication links and the topology of the network. Since the architecture of the transputer fixes the bandwidth and the number of links, the design should concentrate on the network topology.

Graph theory provides a robust and rigorous framework to reason about communication networks. For instance, the delay incurred by a message is proportional to the distance between the two processors involved i.e. the number of processors on the path between the two nodes. Early contributions to distributed computer systems design using graph theory can be found in [Wilkov] and [Harary]. In particular a class of regular graphs known as circulants was introduced by Harary [Harary]. These graphs are node and arc symmetric. Symmetry is a very valuable feature for multi-processors since it simplifies the routing mechanism required to move data and results. Boesh and Wang [Boesch] showed a method for constructing minimum diameter circulants of degree four. The latter are well suited for transputer networks[3]. However, in practice all the nodes in a computer network do not have the same importance. For example, there will be a node whose task is mainly to distribute data and/or processes to the other processors and collect results. This is particularly relevant to computer vision systems where a framestore contains the image or sequence of images to be processed.

A topology was proposed for large transputer networks. It relies on Circulant and De Bruijn graphs (De Bruijn Based Circulants- DBC) and offers very good diameter and average distance characteristiques.

Also, a routing algorithm was designed and implemented in Occam to allow communication between transputers along shortest paths. The routing process is distributed and relies on the simple construction of the DCB graphs.
Practical considerations, besides the routing algorithm, are dealt with in Chapter 8. These include mainly the Occam implementation of a performance monitor. The latter is based on the workings of the transputer's scheduler. This tool proved useful for the optimisation of the parallel computer vision algorithms presented in Chapters 4 to 6.

The advent of the T9000 transputer, and its associated router chip (C104), introduced a different approach to the design of interconnection networks. The Universal Message Passing Computer [May] aims to implement full connectivity in a network through a set of C104 chips. For that a labelling of the nodes in a given network must exist, before communications can proceed along shortest paths. Although such labellings are

[3]Since a transputer has four links.

known to exist for standard topologies (e.g. trees, hypercubes, etc.), this is not the case for arbitrary networks. As a general purpose computer a Universal Message passing Machine is superior to the topology presented here because of ease of programming, portability and easy implementation of emerging standards (MPI). However, applications specifically developed for DBC can take advantage of the simple structure and reconfiguration scheme to implement efficient communications.

The computer vision algorithms considered range from edge detection where two new algorithms were proposed to vector quantisation for image compression, the Fourier descriptors method for shape discrimination, the Hough transform for lines and arbitrary shapes and the maximal clique finding for object recognition.

The Fourier descriptors method, the Hough transform and matching relational structure through the maximal clique algorithm were implemented on various topologies with up to sixteen transputers. Unfortunately, the architecture proposed in Chapter 7 requires a large number of transputers. Therefore, simple 2-D meshes, trees, and linear and bi-linear structures were used for these implementations. However, these structures can all be embedded into the architecture given in Chapter 7.

A formulation of the Fourier descriptors method is proposed which relies on the fast Hartley transform to achieve a high object throughput. The latter is the result of a fast parameterisation of incoming shapes due to the efficiency of the FHT. A parallel implementation of the latter constituted together with the normalisation problem the main aspect of section 6.1. The results show a very fast FD computation which should be more than adequate for applications where a large library of shapes makes the classification task very arduous. In this case a sub-system dedicated to FD computation will allow the remainder to concentrate on classification.

The Hough transform algorithm is used extensively in computer vision mainly because of its immunity to noise. However, its computational complexity dictates that it can only be of practical use in applications which are not time critical. Therefore, an attempt is made at parallelising the algorithm. The results show that large speedups can be achieved by combining a probabilistic approach to edge pixel selection and a parallel implementation.

The maximum clique finding algorithm is a graph theoretical method which consists of detecting the largest all-connected set of nodes in a given graph. This method is used in computer vision in conjunction with the association graph method which builds a

single graph from the two structures to be matched. The problem is NP-complete i.e. no algorithm is known to solve it in time less than an exponential function of the input size (number of nodes in the graph). In this work several attempts were made at formulating the problem in a manner suitable for parallel implementation:

1. block cluster analysis
2. Minimum Distance between all pairs of nodes (MD).
3. transitive orientation.

Modest speedups were noted for a parallel iterative version of a classic algorithm [Bolles]. The three ideas above were analysed and block cluster analysis was found to exhibit inherently serial processing. As far as transitive orientation is concerned it was found that many graphs are not orientable; thus, a closer investigation of the graphs and the contexts which generate them is needed. Therefore, more experimentation and theoretical appraisal are required for the application of these concepts.

This application together with further evaluation of the algorithms of section 6.1 and 6.2 could be the subject of further work. This could take shape as the implementation of complete applications incorporating the algorithms developed here as modules. Another interesting avenue is the integration of different computer vision tasks' implementation into flexible architectures of the type developed in chapter 7 with further consideration to the network optimality and the switching technique.

The transputer offers good characteristics as a building block for a parallel computer vision system . Given that careful care is given to the interconnection network, applications can benefit from the co-operation of several processors in solving a particular task. As some algorithms exhibit very little inherent parallelism or massive global communication needs, it is imperative that a parallel computer vision system allows for reconfiguration. This is shown here to be possible with current technology and is bound to become more so with hardware improvements in the future.

This thesis has shown that graph theoretic methods can help in the design of parallel computers with low diameters and average distances. The network presented in chapter 7 compares favourably with the published literature with regard to these metrics. Besides, it was shown that practical limitations[4] in multi-processor technology can be alleviated through the partition of the PEs into super-nodes and optimisation of the communication characteristiques at two levels. Also high speedups were achieved

[4]e.g. the number of links on a given processing element.

with parallel versions of a selection of low and intermediate level computer vision algorithms. Modest speedups were obtained for the maximal clique algorithm and improvements were proposed. Because the aim was not to solve a particular problem using these algorithms, the results in chapter 6 are concise and only show speedup figures.

APPENDICES

APPENDIX A

Occam and the transputer

This appendix introduces the Occam language and the transputer family of processors and the C004 crossbar switch. First, a brief introduction of the syntax of Occam and program transformation methods is given. The close adherence to the Communicating Sequential Processing theory is noted and the differences mentioned. Second, the transputer family of processors is presented. The basic structure common to all transputers is detailed and the more advanced features of the latest generation are described. The characteristics of the transputer that make it suitable for parallel processing are the four bi-directional serial links -operating in Direct Memory Access (DMA)- and the micro-programmed scheduler which allows several processes to run concurrently on a single processor. The scheduler operation is described in detail as it forms the basis for the algorithm presented in Chapter 5 for measuring utilisation i.e. the ratio of processor time to real time for a particular program.

A-1 Occam

Occam is a very simple programming language. It enables an application to be described as a set of concurrent processes. The latter communicate with each other and with the environment through uni-directional channels. The basic syntax defines three primitive processes:

- v := e evaluate expression e and assign result to variable v
- c ! e evaluate expression e and output to channel c
- c ? v input value from channel c and assign to variable v

Processes are combined using three constructs which differ from equivalent forms in CSP only in notation [Hoare]. These are:

SEQ whose components are executed one after another
PAR whose components are executed together
ALT where only the first component process to be ready is executed

These six forms are the basic elements of any Occam program. The constructs can in turn be seen as processes and may be used as components to other constructs. Serial programs can be expressed with the primitive processes and the SEQ construct. Moreover, IF and WHILE constructs are also provided. The support for concurrent programming takes the form of channel input, channel output and the PAR and ALT constructs. Each Occam channel provides a communication path between two concurrent processes. Communication is synchronised and is effected when processes at both ends of a channel are ready. This frees the programmer from the task of explicitly defining synchronisation operations e.g. hand-shake. This section merely introduces the features of Occam that are of interest for the development of parallel applications, for a full definition of Occam's syntax and semantics see [Occam 2]. The implementation of Occam on the transputer provides a formalism for building parallel processing systems.

Occam provides support for concurrency while the transputers implement parallelism through their links. Therefore, a description of the architecture of the transputer family of processors is given next.

A-2 The transputer

The INMOS transputer family is a range of VLSI[1] circuits used as building blocks for parallel processing systems. Figure 1 displays the basic architecture common to all transputer products.

[1]Very Large Scale Integration

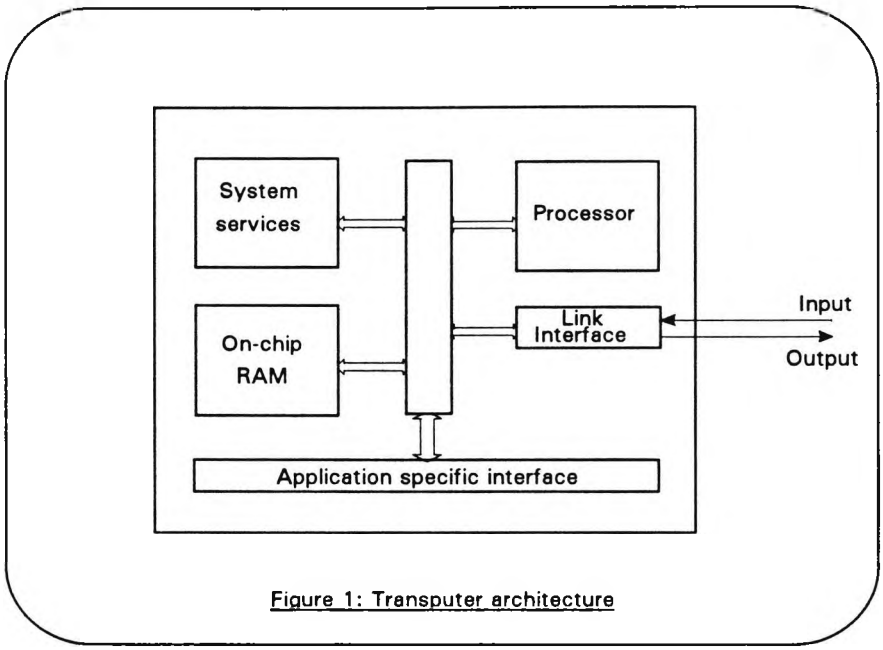

Figure 1: Transputer architecture

The support to concurrency offered by the transputer materialises in the micro-code scheduler. The processor can execute processes at one of two levels of priority, one level (for priority 0) for urgent processes and the other (priority 1) for less urgent processes. Priority 0 processes take precedence and they are executed whenever they become ready. If a priority 0 process becomes ready while a priority 1 process is running, the latter is interrupted.

The processor contains two clock registers, one for each priority. The high priority timer increments every 1 μs and the low priority clock increments every 64 μs. After every 1024 ticks of the high priority clock a timeslice period is said to have ended. When two consecutive timeslice period ends have occurred while the same low priority process has been running, the scheduler will attempt to deschedule the process. This can only happen after a few selected instructions. Then, the process is descheduled and the next waiting process is scheduled. Note that high priority processes are not timesliced and will run until completion or until they have to wait for communication.

The processor maintains two lists of processes that are ready to execute, one for each priority level. Each ready list contains the workspaces (i.e. pointers to locations in memory holding information[2] about the processes) of processes which are ready to be run. A process is started by adding its workspace to the back of the appropriate list. When the current process is descheduled it is also put at the back of the appropriate scheduling list and a new process is taken from the front of the list.

---

[2]This information varies according to the type of process but always includes the instruction pointer.

When a high priority process interrupts a low priority process, the latter is not put at the back of the list. Instead, its current parameters are saved in a special location in memory. When there are no high priority processes able to run the values of the registers are reloaded and the low priority process proceeds. For this reason there can only be one interrupted low priority process at any time.

For more information about Occam and the transputer please refer to the Inmos literature listed in the references section.

A-2.1. The IMS T9000 transputer

The IMS T9000 is the latest member in the transputer family of microprocessors. It integrates a high performance central processing unit (CPU), a 16 Kbyte cache, communications system and other support functions on a single chip. The main functional blocks of the IMS T9000 are the CPU, the hierarchical memory system, the communications system, the multiple internal buses and the system services unit. The function of each of these is outlined below.

The IMS T9000 CPU contains a 32 bit arithmetic and logic unit (ALU) and a 64 bit floating point unit (FPU). The CPU also includes instructions for byte and half word operations. The CPU uses 32 bit linear addressing and the IMS T9000 is binary compatible with previous transputers. In particular it implements the instruction set of the IMS T805 with many additions.

One of the major improvements over the T800, besides better error correction, is the ability of the T9000 to run code in protected mode. In this mode all memory accesses are made through a memory management unit which checks and translates addresses before using them to address the memory system. Further, only a subset of the full instruction set may be executed, preventing protected code from executing privileged instructions.

Hierarchical memory system

The IMS T9000 includes a 16 Kbyte unified cache to provide single cycle access to instructions and data. The cache provides a peak bandwidth of 200 Mwords/s. The CPU also includes another small cache for the most frequently used local variables of a program which provides another 150 Mwords/s of memory bandwidth.

There are four independent sets of memory control signals simplifying the use of different device types in the same system. The memory can interface to 8, 16, 32 or 64 bit wide devices. The maximum data transfer rate across the memory interface is 50 Mwords/s.

Communications system

To support interprocessor communications, there is a complete communications subsystem on chip. This includes four 100 Mbits/s full-duplex, serial communication links each with its own pair of direct memory access (DMA) channels. The links can be directly connected between transputers with no external buffering or other glue logic. A communications processor, which manages all link communications, operates concurrently with the main CPU so that data transfers do not adversely affect CPU operation.

Two additional links are provided for system control and monitoring. Initialisation and booting of the processor can optionally be done through these links. The communications subsystem also includes four `Event' channels. As well as acting as interrupt inputs, these can be used, as inputs or outputs, for more general synchronization and signalling.

Multiple internal buses

To support the high degree of concurrent operation on the IMS T9000, and to maintain the high internal data rates required, there are four sets of 32 bit address and data buses internally. These provide multi-port access to the on-chip cache from the various functional units of the IMS T9000.

System services

The system services section provides all the general facilities necessary for the operation of the transputer. This includes the power and ground connections, and the clock input (5 MHz). Other important connections are a capacitor, which is required for the on-chip phase locked loops which generate all the internal high frequency clocks, and the processor speed select pins which can be used to select the frequency of the internal clocks (up to the maximum speed for a particular device). There is also a reset input - however, as the IMS T9000 includes on-chip power-on reset circuitry, external reset logic may not be required in an embedded control application.

Pipelined, superscalar implementation

To increase the execution rate of the transputer instruction set, the IMS T9000 is able to issue several instructions per cycle. A superscalar micro-architecture was designed which implements the same high level architecture and instruction set as the IMS T805 but with much higher performance.
The details of the IMS T9000 pipeline are transparent to the programmer. The processor appears to be the simple transputer architecture described above and straightforward code written for that programming model will get nearly the best performance out of the processor. An optimising compiler for the IMS T9000 can, of course, generate more efficient code if the details of the internal architecture are taken into account.

The pipeline

Instructions are executed in a five stage pipeline: the first can fetch two local variables; the second can perform two address calculations, for accessing non-local or subscripted variables; the third stage can load two non-local variables; the next can perform an ALU or FPU operation; and the final stage can do a conditional jump or write.
A conventional pipeline is designed to allow several instructions to be executed simultaneously; different parts of each instruction being handled in different stages of the pipeline. In order to allow multiple instructions to be issued per cycle (as well as multiple instructions being executed in each cycle) the IMS T9000 does not simply send a sequence of instructions through the pipeline but has hardware which assembles groups of instructions from the instruction stream. These groups are chosen to make the best use of the available hardware and one group can be sent through the pipeline every cycle. Instructions are put into groups in the order that they arrive at the CPU; dependencies within the group are handled automatically by the pipeline.

The grouper can be thought of as a hardware optimiser; it recognises commonly occurring code sequences that the processor can execute effectively. The design of the grouping mechanism and the pipeline is based on analysis of the code typically generated by high level language compilers.

Grouping of instructions

The grouping of instructions takes advantage of the high degree of concurrency and multiple buses in the processor. For example, both caches are multi-ported and can each support two reads by the CPU simultaneously.

Since the processor can fetch one word, containing four bytes of instructions and data, in each cycle it is possible to achieve a continuous execution rate of four instructions per cycle (200 MIPS). However, if any of the instructions require more than one cycle to execute, then the instruction fetch mechanism can continue to fetch instructions so that larger groups can be built up. Up to 8 instructions can be put into one group and there may be five groups in the pipeline at any time.

Improvements over IMS T800

In addition to executing several instructions each cycle, the number of cycles required to perform many arithmetic and logical operations has been reduced from previous transputers by adding extra hardware. This, combined with the faster clock speed and the new micro-architecture, means a ten-fold increase in speed over the IMS T805. In addition there is improved support for error handling, and protecting code and data from the errant behavior of a program. The IMS T9000 provides better access to the transputers scheduling mechanism, making it easier to implement software kernels for a particular processing model.

Hierarchical memory system

The IMS T9000 has a complete, hierarchical memory system providing fast and efficient access to data and instructions. There are two separate caches on chip, a general purpose unified (code and data) cache and a small cache for local variables. These caches can provide fast, multi-ported access to data because they are on chip. They also reduce the number and frequency of accesses to external memory, allowing lower cost, slower devices to be used without degrading performance. Finally, because the majority of external memory accesses will be cache refills (and therefore multiple word reads and writes) fast memory access methods, such as page mode, can be used.

Main cache

The main cache consists of four independent banks, each containing 256 lines. Each line holds data from four consecutive words (16 bytes) in memory. An access can be made to every bank on every cycle which, with the multiple internal buses, means there is a very high bandwidth between the cache and different functional units within the IMS T9000.

The main function of the C104 routing chip is as follows. It uses the protocol of the T9000 to locate the message destination, derives the output link and immediately starts forwarding the packet (provided that the output link is not in use). Therefore, it introduces an efficient queuing system since the packet size is relatively small (maximum 32 data bytes).

APPENDIX B

Complexity theory terminology

In this thesis the order notation used is that introduced by Knuth [Knuth 1976]. There are three symbols O(), Ω() and Θ().

Assume that f and g are two functions over the set of natural numbers. Then $O(f(n))$ is an upper bound on the order of a particular function $h(n)$ and is defined as follows: It is the set of all $g(n)$ such that there exist positive constants c and n_0 so that $g(n) < c.f(n)$ for all $n > n_0$. Therefore, n, $23n^2+7n+12$ and $3n^2$ are all $O(n^2)$, this can be shown by solving the above inequality for c and n_0 and noting that the predominant term for high values of n is the highest power of n.

Similarly, $\Omega(f(n))$ defines a lower bound on the order of functions: It is the set of all $g(n)$ so that there exist positive constants c and n_0 satisfying $g(n) > c.f(n)$ for all $n > n_0$.

Finally, $\Theta(f(n))$ is defined as the exact order of a function, it is the set of all $g(n)$ such that $g(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Another aspect of the definition and classification of algorithms relies on the nature of $f(n)$. Let P be the class of all problems which can be solved by a polynomial-time algorithm. There are a large number of problems for which this is not the case. Some of these can be solved in polynomial time by a non-deterministic algorithm[3]. NP denotes the class of all such problems. A problem is NP-hard if a deterministic polynomial-time algorithm for its solution can be used to find the same for all problems in NP. Finally, an NP-hard problem in NP is called NP-complete.

---

[3]The Shortest distance amongst all vertices method of section 6.3 is an attempt to find such an algorithm.

APPENDIX C

Shortest path between all pairs of vertices

This appendix introduces the algorithm due to Floyd [Swamy] for computing the shortest paths between all pairs of vertices in a graph. The original algorithm was designed for directed graphs with length associated with their edges. Besides, it is more general in that the succession of nodes in the shortest paths are also provided. Here a simple version for undirected graphs is given. The formulation requires that the absence of an edge from the is represented by an infinite entry at the corresponding location in the nxn adjacency matrix. Therefore, the null off-diagonal elements of the matrix are replaced by ∞. The diagonal elements are set to 0. The matrix W thus obtained is the initial condition of Floyd's algorithm, $W=W^0$. The sequence of matrices $W^1, W^2, \ldots W^n$ is constructed as follows:

$$w_{ij}^k = \min \{w_{ij}^{k-1}, w_{ik}^{k-1}+w_{kj}^{k-1}\} \text{ where } W^l=[w_{ij}^l] \quad 0<l\leq n.$$

On completion the entry w_{ij}^n in W^n will give the length of the shortest path from node i to node j. The following algorithm implements the method.

- **Step 1** W is the starting condition i.e. the modified adjacency matrix.
- **Step 2** set k=0.
- **Step 3** set k=k+1, for all $i\neq k$ and $w_{ik}\neq\infty$, and all $j\neq k$ and $w_{kj}\neq\infty$:

    1. Set $m=\min\{w_{ij}, w_{ik}+w_{kj}\}$
    2. If $m < w_{ij}$ set $w_{ij}=m$.

- **Step 4** If $k<n$ go to step 3 else $W=[w_{ij}]$ gives all the shortest paths.

The above algorithm is efficient and only involves tests and scalar additions. The construction of small meshes of minimum diameter (thus computer networks) based on this algorithm can done as follows. For simplicity it will be assumed that the number of processors in the network is $n=p^2$ where p=1,2,...For other values of n the aspect ratio of the polygon should be made as close as possible to 1. This assertion stems from simple Euclidean geometry whereby a square has the minimum perimeter over all

rectangles with the same area. The area of a mesh connected network represents the number of processors and the small perimeter ensures that distances between side processors are not too large. Having set-up the mesh connections, the wrap around edges are added from one side to the other following a set of permutations after which the above algorithm is run.

Note the purpose here is to reduce diameter of the graph, therefore, only one of the two co-ordinates needs be permuted. The other should implement the identity permutation. For example, the top left node is connected to the bottom left node. This technique is found in many classic configurations including mesh connected circulants.

APPENDIX D

Selected Code Sections 6.1 and 6.2

This appendix presents code fragments which implement the algorithms of sections 6.1 and 6.2. E-1 shows a procedure which takes a sequence of numbers and returns its Hartley transform. The listing E-2 takes data in the form of a list of candidate boundary points and consults the R-table of a parametrised boundary at different scales and rotations to accumulate records of the reference point locations. The latter are returned to the farm controller which in turn forms the Hough space.

D-1 Hartley transform

```
PROC ([]REAL32 x, VAL []REAL32 C, S, VAL INT N, M, n, VAL []INT seed)
  VAL INT Nby2  IS N/2:
 VAL INT Nby4  IS N/4:
 SEQ
  permute (x, M, 2, n, seed)
  -- First stage
  INT U, L:
  REAL32 temp:
  SEQ
   U := 0
   SEQ i = 0 FOR Nby2  -- Two points per FHT
    SEQ
     L := U + 1  -- Lower pt.
     temp := x[L]
     x[L] := x[U] - temp
     x[U] := x[U] + temp
     U   := U + 2 -- Two nodes per FHT
  -- Second stage
  INT U1, U2, L1, L2:
  REAL32 temp:
  SEQ
   U1 := 0
   SEQ i = 0 FOR Nby4  -- Two points per FHT
```

```
    SEQ
      U2 := U1 + 1
      L1 := U1 + 2
      L2 := U1 + 3
      temp := x[L1]
      x[L1] := x[U1] - temp
      x[U1] := x[U1] + temp
      temp  := x[L2]
      x[L2] := x[U2] - temp
      x[U2] := x[U2] + temp
      U1    := U1 + 4 -- Four nodes per FHT
  -- Other stages
  VAL INT Nby8    IS N/8:
  VAL INT MMin2 IS M-2:
  INT fhts, n.p.fht, p.p.fht, p.p.fhtby2:
  INT theta:  -- Angle of rotation step
  SEQ
    -- Init values for this stage
    fhts := Nby8   -- fhts = N/2, N/4, thus fhts = N/8 in third stage
    n.p.fht    := 1  -- In third stage
    p.p.fht    := 8 -- In third stage
    p.p.fhtby2  := 4
    theta    := Nby8
    SEQ stage = 3 FOR MMin2
      INT first.Node.fht:
      INT eta:
      SEQ
        n.p.fht   := n.p.fht + n.p.fht
        first.Node.fht := 0
        SEQ fht = 0 FOR fhts
          SEQ
            -- First node
            INT U, L:
            REAL32 temp:
            SEQ
              U := first.Node.fht
              L := first.Node.fht + p.p.fhtby2
              temp     := x[L]
```

```
  x[L] := x[U] - temp
  x[U] := x[U] + temp
-- General nodes
eta := theta  -- Initial value for eta
SEQ
  SEQ U1 = (first.Node.fht + 1) FOR (n.p.fht - 1)
    INT U2, L1, L2, offset:
    SEQ
      offset  := first.Node.fht + (fht TIMES p.p.fht)
      U2 := (p.p.fhtby2 - U1) + offset
      L1 :=  p.p.fhtby2 + U1
      L2 := U2 + p.p.fhtby2
      -- Butterfly
      REAL32 temp1, temp2, temp3, temp4:
      REAL32 A, B:
      SEQ
        temp1 := x[L1] * C[eta]
        temp2 := x[L2] * S[eta]
        temp3 := x[L1] * S[eta]
        temp4 := x[L2] * C[eta]
        A := temp1 + temp2 -- x[N/2+k]C(T) + x[N-k]S
        B := temp3 - temp4 -- x[N/2+k]S(T) - x[N-k]C
        x[L2] := x[U2] - B
        x[L1] := x[U1] - A
        x[U2] := x[U2] + B
        x[U1] := x[U1] + A
      eta := eta + theta
-- Last node
INT U, L:
REAL32 temp:
SEQ
  U := first.Node.fht + (p.p.fhtby2 >> 1)
  L := U + p.p.fhtby2
  SEQ
    temp     := x[L]
    x[L] := x[U] - temp
    x[U] := x[U] + temp
first.Node.fht := first.Node.fht + p.p.fht
```

```
      fhts   := (fhts >> 1)  -- One DFT in last stage
      p.p.fhtby2 := p.p.fht  -- In prev. stage
      p.p.fht    := p.p.fht + p.p.fht
      theta := (theta >> 1)
:
```

D-2 Generalised Hough transform

```
-- Data is presented to the following process in the form of: x,y,dx,dy the positions and gradient magnitudes respectively.
IF
 (dx <> 0)
  angle.in.radians := ATAN ((REAL32 ROUND dy)/(REAL32 ROUND dx)) -- result
 TRUE
  angle.in.radians := PiBy2
theta.edge := INT ROUND (angle.in.radians*57.3 (REAL32))
theta.edge := theta.edge + 90  -- get to appropriate place in R-table
-- check for overflow
IF
 theta.edge = 180
  theta.edge := 0
 TRUE
  SKIP
theta.edge := (theta.edge + r.angle) \ 180
SEQ i = 0 FOR (r.table.position[theta.edge]>>1)
 INT   r, theta, new.dx, new.dy, tempx, tempy :
 REAL32 r.real, theta.real, sin.h, cos.h :
 SEQ
  r     := r.table[theta.edge][(i<<1)  ]
  theta := r.table[theta.edge][(i<<1)+1]  -- theta
 -- work out orientation to ref point
 theta := theta - r.angle
 IF
  (theta < (-90))
   theta := theta + 180
  TRUE
   SKIP
  -- work out new r
 r.real := REAL32 ROUND r              -- r (real)
```

```
-- r.real := r.real * (REAL32 ROUND r.scale)  -- new scale
-- now work out new dx and dy
theta..real := REAL32 ROUND theta.  -- theta (real)
sin.h := SIN( theta.real/57.3 (REAL32) )
cos.h := COS( theta.real/57.3 (REAL32) )
new.dx := INT ROUND (cos.h*r.real)  -- new values of dx and dy
new.dy := INT ROUND (sin.h*r.real)
--  new x and y for Hough space
IF
  new.dx < 0
    new.dx := -new.dx
  TRUE
    SKIP
IF
  new.dy < 0
    new.dy := -new.dy
  TRUE
    SKIP
IF
  theta < 0
    tempx := x + new.dx
  TRUE
    tempx := x - new.dx
tempy := y - new.dy
IF
 -- Within Hough space
 (tempx>0) AND (tempx<255) AND (tempy>0) AND (tempy<255)
   INT current:
   SEQ
     current:= no.points[pos]
     table.of.results[pos][current] := BYTE tempx
     table.of.results[pos][current+1] := BYTE tempy
     no.points[pos] := current+ 2
 TRUE
   SKIP
```

References

[Ambler] Ambler, A.P, Barrow, H.G., Brown, C.M., Burstall, R.M. and Popplestone, R.J. "A versatile Computer controlled assembly system." Proc. 3rd Int. Joint Conf. Art. Intell., Stanford, CA, pp 298-307, 1973.

[Arsac] Arsac, J. "Foundations of Programming." Academic Press, 1985.

[Ashkar] Ashkar, G.P. and Modestino, J.W. "The Contour extraction problem with biomedical applications." CGIP, 7, pp 331-355, 1978.

[Astfalk] Astfalk, G. "Convex's view of tflops computing." Parallel Computing and Transputer Applications, Part I, pp 51-61, Valero *et al* eds, CIMNE, Barcelona, 1992.

[Ballard] Ballard, D.H. and Brown, C.M. "Computer vision." Prentice-Hall, New Jersey, 1982.

[Ballard 2] Ballard, D.H. Generalizing the Hough Transform to detect arbitrary shapes." Pattern recognition 13, 2, pp 111-22, 1981.

[Ballard 3] Ballard, D.H. "Hierarchic detection of tumors in chest radigraphs." Verlag, 1976.

[Baude] Baude, F., Carre, P. and Vidal-Naquet, G. " Topologies for Large Transputer Networks: Theoretical Aspects and Experimental Approach." Proc. OUG-10, A. Bakkers ed, pp 178-197, 1989.

[Beguelin] Beguelin, A "PVM and HeNCE " Technical Report, University of Tenesse, 1990.

[Belhaire] Belhaire E., Garda P., Bernard T., Devos F. and Zavidovique B. "A Diffusion Based Edge Detector." 22th Asilomar Conf Sign. Comp., pp 315-19, 1988.

[Bevide] Bevide, R., Herrada, E., Balcazar, L. and Labarta, J. "Optimized Mesh-Connected Networks for SIMD and MIMD Architecture." J. ACM, pp 163-170, 1987.

[Binford] Binford, T.O. "Survey of Model-Based Image Analysis Systems." The international Journal of Robotics Research, pp 18-64, 1982.

[Boesch] Boesch, F.T. and Wang, J. "Reliable Circulant Networks with Minimum Transmission Delay." IEEE trans. Circ.Syst., CAS-32, No 12, pp 1286-91, 1985.

[Bolles] Bolles "Robust feature matching through maximal cliques." SPIE, 182, pp 140-9, 1979.

[Bracewell] Bracewell, R.N. "The fast Hartley transform." Proc. IEEE, 72, pp 1010-8, 1984.

[Brice] Brice, C.R. and Fennema, C.L. "Scene analysis using regions." Artificial Intelligence, 1, No 3 pp 205-226, 1970.

[Brillault] Brillault-O'Mahony, B. "A probabilistic Approachh to 3d Interpretation of Monocular Images." PhD thesis, City University, London, 1992.

[Canny] Canny.J.F. "A computational approach to edge detection", IEEE Trans. PAMI, 8, no.6, pp 679-97, 1986.

[Chen] Chen Z. and Ho, S.Y. "3D aircraft recognition with fast library search." Pattern Recognition, 24, No 5, pp 375-90, 1991.

[Claxton] Claxton, P.R. and Kwok, E.K.W. "The use of colour to segment and label images", Proc. 3rd Alvey Vision Conference, AVC87, 15-17 Sept., Cambridge, UK, p. 295-302, 1987.

[Coleman] Coleman, G.B. and Andrews, H.C. "Image Segmentation by Clustering." Proc. IEEE, pp 773-785, 1979.

[Cooley] Cooley, J.W. and Tukey, J.W. "An Algorithm for the Machine Calculation of Complex Fourier Series." Math. Comput., 19, pp 297-301, 1965.

[Davies 1] Davies, E.R. "A new parametrisation of the straight line and its apllication for the optimal detection of objects with straight edges." Pattern Recognition Letters, 6, pp 9-14, 1987.
[Davis] Davis, L.S., Johns, S.A. and Aggarwal, J.K. "Texture Analysis Using Generalized Co-Occurence Matrices." PAMI, 1, No 3, pp 251-9, 1979.
[Debbage] Debbage, M. and Hill, M. "The Virtual Channel Router " ESPRIT P2701 PU MA, PUMA /035/VCR2/ADVT, Nov, 1992.
[De Bruijn] De Bruijn, N.G. "Some machines defined by directed graphs." Theoretical Computer Science, 32, pp 309-19, 1984.
[Deriche] Deriche, R. "Optimal Edge Detection using Recursive Filtering." Proc. Int. Conf. Comp. Vision, London, 1987.
[Duda 1] Duda, R.O. and Hart, P.E. "Use of the Hough transformation to detect lines and curves in pictures." Commun. ACM, 15, No 1, pp 11-5, 1972.
[Duda] Duda, R.O. and Hart, P.E. "Pattern Classification and Scene Analysis." John Wiley and Sons, New York, 1973.
[Duato] Duato, J. and Garcia, J.M., "Evaluating the cost of the dynamic reconfiguration of a multicomputer network" Actas de la XVIII Conferencia Latinoamericana de Informatica PANEL 92, pp 523-30, Las Palmas Gran Canaria, September, 1992.
[Ellis] Ellis, T.J., Hung, T.W.R. and Omarouayache, S. "Structural Elements in colour images: A parallel Approach." Proc. of the third International Conference on the application of transputers, pp 536-41, Glasgow, 1991.
[Feng] Feng, T.-Y., "Some Characteristics of Associative/Parallel Processing", Proc. of the 1972 Sagamore Computer Conference", pp 5-16, August, 1972.
[Floyd] Floyd, R.W. "Algorithm 97: Shortest Path." Comm. ACM, 5, 345, 1962.
[Flynn] Flynn, M.J. "Very high speed computing systems." Proceedings of the IEEE 54, 1966, 12, pp. 1901-9.
[Freeman] Freeman, H. "Computer processing of line drawing images." Computer Surveys, 6, No 1, pp 57-98, 1974.
[Frei] Frei, W. and Chen, C.C. "Fast Boundary Detection: A Generalization and a New Algorithm." IEEE trans. Comp., c-26, No 10, 1977.
[Garcia] Garcia, J.M. and Duato, J. " An Algorithm for Dynamic Reconfiguration of a Multicomputer Network." Proc. 3rd IEEE Symp. Parallel and Distributed Processing, pp 848-55, Dallas, 1991.
[Gonzalez] Gonzalez, R.C. and Wintz, P. " Digital Image Processing." Addison-Wesley, Reading, 1987.
[Gottlieb] Gottlieb, A. Architectures for parallel supercomputing. Parallel Computing and Transputer Applications, Part I, pp 39-47, Valero *et al* eds, CIMNE, Barcelona, 1992.
[Gupta] Gupta, J.N. and Wintz, P.A. "A Boundary Finding Algorithm and its Applications." IEEE trans. Circuits Sys., Cas-22, No 4, pp 351-62, 1975.
[Händler] Händler, W. The impact of classification schemes on computer architecture. Proceedings of the Int. Conf. on Parallel Processing, August 1977, pp 7-15, IEEE, New York.
[Harary] Harary, F. "Graph Theory." Addison-Wiley, Reading, 1969.
[Hart] Hart, E. "Transim: Prototyping Parallel Algorithms", University of Westminster Press, London, 1994.
[Hoare] Hoare, C.A.R. "Communicating Sequential Processes.", Prentice-Hall, 1985.

[Horowitz] Horowitz, S.L. and Pavlidis, T.P. "Picture segmentation by a tree traversal algorithm." J. ACM, 23, No 2, pp 368-88, 1976.
[INMOS 1] The Transputer Applications Notebook, Systems and Performance, Inmos, 1989.
[Inmos1] Inmos. "IMS T800 Architecture." technical note 6.
[Johnston] Johnston, H.C. "Cliques of a graph - large or small." Queen's University Belfast, 1975.
[Jones 1] Jones, G. "Measuring the busyness of a transputer." WoTUG Newsletter, No 12, Jan. 1990.
[Jones] Jones, P. and Murta, A. "Practical experience of run-time link reconfiguration in a multi-transputer machine." Concurrency: Practice and Experience, 1(2), pp 171-189, 1989.
[Khoros] The Khoros manual pages, University of New Mexico, 1991.
[Kimme] Kimme, C., Ballard, D.H. and Sklansky, J. "Finding circles by an array of accumulators." Commun. ACM, 18, No 2, pp 120-2, 1975.
[Kiryati] Kiryati, N., Eldar, Y. and Bruckstein, A.M. "A probabilistic Hough transform." Pattern Recognition, 24, No 4, 1991.
[Kittler] Kittler, J. "On the accuracy of the Sobel Edge detetector." Image Vision Comput., 1, pp 37-42, 1983.
[Köninger] Köninger, R.K., "MPP Directions at Cray Research Inc.", Parallel Computing and Transputer applications, Part 2, Valero *et al* eds. pp 80-90, CIMNE & IOS Barcelona 1992
[Kuhl] Kuhl, F.P. and Giardina, C.R. "Elliptic Fourier feature of a closed contour." Comp. Graphics Image Processing, 18, pp 236-58, 1982.
[Langhammer] Langhammer, F. "Second generation and teraflops parallel computers." Parallel Computing and Transputer Applications, Part I, pp 62-79, Valero *et al* eds, CIMNE, Barcelona, 1992.
[Lau] Lau, S.W. and Lau, F.C.M. "A Simple Cut-Through Router." WoTUG Newsletter, No 15, July 1991.
[Laws] Laws, K.I. "Rapid texture identification." SPIE, 238, pp 376-380, 1980.
[Leavers] Leavers, V.F., Ben-Tzvi, D. and Sandler, M.B. "A dynamic combinatorial Hough transform for straight lines and circles." Proc. 5th Alvey vision conference, Univ. Reading, pp 163-8, 1989.
[Lester] Lester, J.M., Williams, H.A., Weintraub, B.A. and Brenner, J.F. "Two graph searching techniques for boundary finding in white blood cells." Comp. Bio. Medecine, 8, pp 293-308, 1978.
[Lipovski] Lipovski, G.J. and Miroslaw, M., "Parallel Computing Theory and Comparisons", John Wiley & Sons, New York, 1987.
[Liu] Liu, C.L. "Introduction to Combinatorial Mathematics", McGraw-Hill, New York, 1968.
[Lowe] Lowe, D.G. "Three-Dimensional Object Recognition from Single Two-dimensional Images." Artificial Intelligence, 31, pp 355-395, 1987.
[Marr] Marr, D.C. and Hildreth, E. "Theory of edge detection." Proc. Royal Society London, Vol B207, pp 187-217, 1980.
[Martelli] Martelli, A. " An application of heuristic search methods to edge and contour detection." Comm. ACM, 19, No 2, pp 73-83, 1976.
[May] May, M.D., Thompson, P.W. and Welch, P.H., "Networks, Routers and Transputers: Function, Performance, and Applications", Inmos Limited, Bristol, 1993.

[Mazzeo] Mazzeo, A. Mazzocca, N. and Villano, U. "A transputer architecture for high-speed 1D- and 2D-FFT computations." Parallel Computing and Transputer Applications, Part I, pp 317-26, Valero *et al* eds, CIMNE, Barcelona, 1992.
[Mitchell] Mitchell, O.R., Myers, C.R. and Boyne, W. "A max-min Measure for Image Texture Analysis." IEEE trans. Comp., pp 408-414, 1977.
[Nevatia] Nevatia.R. "A color edge detector and its use in scene segmentation", IEEE Trans. SMC, 7, No 11, pp 820-26, 1977.
[Nicole] Nicole, D.A., Lloyd, E.K. and Ward, J.S. "Switching Networks for Transputer Links." Proc. 8th Tech Meeting of OUG, Kerridge. J ed, 1988.
[Nilsson] Nilsson, N.J. "Priciples of Artificial Intelligence." Paolo Alto, CA, 1980.
[Omar 1] Omarouayache, S., Ngwa-Ndifor, J. and Ellis, T.J. "Vector quantisation : a transputer implementation" Proc. IEE International Conference on Image Processing and its applications, Maastricht, 1992.
[Omar 2] Omarouayache, S., Mylonas, S.A, ELLIS, T.J, and R.A. Comley "Transputer implementation of adaptive noise cancelling in digital images", Parallel Computing and Transputer applications, Part 2, Valero *et al* eds. pp 964-73, CIMNE & IOS Barcelona 1992.
[Omar] Omarouayache, S. progress report on detection of brain tumors, Dept. of Medical Oncology, Charing Cross Hospital, London, 1992.
[Ore] Ore, O. "Graphs and Their Uses." Random House, New York, 1963.
[Orfanidis] Orfanidis, S. - "Optimum Signal Processing - An Introduction", 2nd Ed., McGraw-Hill, pp 405-34.
[Peel] Peel, R.M.A. "Virtual Cut-Through Routing." WoTUG Newsletter, No 16, Jan. 1992.
[Persoon] Persoon, E. and Fu, K.S. "Shape discrimination using Fourier Descripors." IEEE Trans. Syst. Man Cyber., SMC-7, 170-9, 1977.
[Pingle] Pingle, K.K. "Visual perception by computer." Automatic Interpretation and Classification of Images, A Grasselli Ed., New York, Academic Press, pp 277-84.
[Pneuli] Pneuli, A., Lempel, A. and Even, S. "Transitive Orientation og graphs and identification of permutation graphs." Canad J. Math., 23, pp 160-75, 1971.
[Quinn] Quinn, M.J. "Designing Efficient Algorithms for Parallel Computers", McGraw-Hill, New York, 1987.
[Ramamurthi] Ramamurthi. & Gersho. "Classified vector quantization of images" IEEE Trans. COMM., 34, No 11, pp 1105-15, 1986.
[Redfern] Redfern, S. "Implementing data structures and recursion in Occam." Inmos, 1988.
[Rice] Rice, S.O. "Mathematical Analysis of Random Noise." Bell Syst. Tech. J., 24, pp 46-156, 1945.
[Robinson 1] Robinson, G. "Color Edge Detection", Proc. SPIE Symp. Advances
[Robinson] Robinson, J.T. "Analysis of asynchronous multi-processor algorithms with application to sorting." Proc. Int. Conf. Parallel Processing IEEE, pp 128-35, New York, 1977.
[Rosenfeld 2] Rosenfeld, A. "Picture Processing by Computer." New York Academic Press, 1969.
[Rosenfeld] Rosenfeld A., Ornelas J. and Hung Y, " Hough Transform Algorithms for Mesh-Connected SIMD Parallel Processors." Computer Vision, Graphics and Image Processing, 41, pp 293-305, 1988.
[Ruff] Ruff, B.P.D. "A pipelined Architecture for the Canny Edge Detector." Alvey Programme IKBS 025.

[Skillicorn] Skillicorn, D.B. "A Taxonomy for Computer Architectures", Computer, Vol. 21, No 11, pp46-57, November, 1988.
[Gropp] Gropp, W., Lusk, E. and Skjellum, A. "Using MPI - Portable Parallel Programming with the Message-Passing Interface". MIT Press, 1994.
[Swamy] Swamy, M.N.S. and Thulasiraman, K. "Graphs, Networks and Algorithms." John Wiley and Sons, New York, 1981.
[Tai] Tai, A., Kittler, J., Petrou, M. and Windeatt, T. "Vanishing Point Detection." BMVC, pp 109-116, 1992.
[Temperley] Temperley, H.N.V. "Graph theory and applications", Ellis Horwood Limited, Chichester, 1981.
[Tucker 2] Tucker, L.W., Feynman, C.R. and Fritzsche, D.M. "Object Recognition Using the Connection Machine." Proc. IEEE Conf. Computer Vision and pattern Recognition, pp 871-7, June 1988
[Tucker] Tucker, L.W. and Robertson, G.G. "Architecture and Applications of the Connection Machine." Computer, Vol. 15, No 1, pp 26-38, 1988.
[Tutte] Tutte, W.T. "Connectivity in graphs." London, Oxford Univ. Press, 1966.
[Wallace] Wallace, T.P. and Wintz, P., "An efficient three-dimensional aircraft recognition algorithm using normalized Fourier descriptors." Comp. Graphics Image Process., 13, pp 99-126, 1980.
[Waltz] Waltz, D.L. "Applications of The connection Machine." Computer,Vol. 20, No 1, pp 85-97, 1987.
[Widrow 1] Widrow, B. *et al.* - Adaptive noise cancelling: Principles and Applications, Proc. IEEE, 63, *No 12*, pp 1692-1716, 1975.
[Widrow 2] Widrow, B. *et al.* - Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter, Proc. IEEE, 64, No 8, August 1976, pp 1151-62.
[Wilkov 1] Wilkov, R.S. "Construction of maximally reliable communication networks with minimum transmission delay." Proc. IEEE Int. Conf. Comm., 6, pp 4210-5,1970.
[Wilkov 2] Wilkov, R.S. "On the design of high speed communication networks." 5th annu. Conf. Info. Sci. Syst., Princeton, 1971.
[Wilkov] Wilkov, R.S. "Analysis and Design of Reliable Computer Networks." IEEE trans. Comm., COM-20, No 3, pp 660-78, 1972.
[Wu] Wu, S., Abel, J.F. and Greenberg, D.P. "An interactive computer graphics approach to surface representation." Comm. ACM, 20, No 10, pp 703-11, 1977.
[Yuen] Yuen, S.Y.K. "Connective Hough Transform." Proc. BMVC, Mowforth, P. ed, pp 127-33, 1991.
[Zahn] Zahn, C.T. and Roskies, R.Z. "Fourier Descriptors for Plane Closed Curves." IEEE trans. Comp., C-21, pp 269-81, 1972.
in Image Transmission Techniques, no. 87.