PARALLEL PROCESSOR IMPLEMENTATION IN COMPUTERIZED TOMOGRAPHY USING TRANSPUTERS

A THESIS PRESENTED IN PARTIAL FUFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF TECHNOLOGY IN PRODUCTION TECHNOLOGY AT MASSEY UNIVERSITY

WENJU WU

1993

Abstract

Image reconstruction by computerized tomography provides a nonintrusive method of imaging the internal structure of objects. From measurements of radiation (e.g. X-rays or gamma rays) passed through an object, it is possible to observe the internal structure. The reconstruction process is computationally intensive and requires imaginative parallel processing algorithms to attain 'real-time' performance. The Inmos transputer makes parallel processing algorithms both feasible and relatively straight forward. In this thesis, a modification to the backprojection algorithm is introduced in order to improve the speed of the implementation. Work carried out has involved evaluating how these algorithms (convolution, backprojection and interpolation) can be used in multiprocessor concurrent architecture to obtain rapid image reconstruction. Several suitable transputer network structures have been advanced to simulate the image reconstruction. The reconstruction time is decreased very greatly and the image reconstruction result is good.

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Dr. Bob Chaplin, for his guidance throughout this project.

I would like to thank Prof Bob Hodgson and Dr Bob Chaplin for their help and care during the course of my study, Dr Roger Browne for his help throughout the project and Mr White Page for his help in image research.

My thanks also extend to Dr Paul Austin, Dr Huub Bakker, Mr Ralph Pugmire, Dr Clive Marsh, Mr Merv Foot and all staff in the Production Technology Department of Massey University for their help and friendship.

I wish to gratefully acknowledge Mr Phil Long for all his help during my research period.

I also like to mention the enjoyable company and moments provided by the postgraduate students of Production Technology.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures viii

Chapter 1 Introduction to X-Ray Computerized Tomography 1

1.1 Digital Image Processing 1

1.1.1 The elements of digital image processing 2

1.1.2 Basic classes of digital image processing 3

1.2 The Fundamentals of Computerized Tomography 7

1.2.1 Theoretical background for image reconstruction 9

1.2.1.1. The N-Dimensional Continuous Fourier Transform (CFT) 9

1.2.1.2. The N-Dimensional Discrete Fourier Transform (DFT) 11

1.2.1.3. Projections 14

1.2.1.4. The Projection-Slice Theorem 15

1.2.1.5. The Basis for Reconstruction from Projections 17

1.2.1.6. Reduction of the Dimensionality of the Reconstruction Problem 18

Chapter 2 Introduction to Parallel Processing and Transputers 19

2.1 The Principles of Parallel Processing Systems 19

2.2 Description of Parallel Processes 22

2.2.1 Initiation and termination mechanisms 23

2.2.2 Synchronization mechanism 23

2.2.3 Protection mechanism 24

2.2.4 Communication mechanism 24

2.3 The Transputers 25

2.3.1 Overview 25

2.3.2 Occam 27

2.3.3 Communication channels 30

Chapter 3 Reconstruction Algorithms for Parallel and Fan Beam Projections 32

3.1 Reconstruction Algorithm for Parallel Projections 32

3.1.1 The idea 32

3.1.2 Theory 35

3.2 Reconstruction Algorithm for Fan Projections 40

3.3 Modified Backprojection Algorithm 44

Chapter 4 Parallel Processor Implementation in Computerized Tomography using Transputers 52

4.1 Analysis of the Reconstruction Algorithm 52

4.2 IMS T414 Transputer 57

4.2.1 Pin designations 59

4.2.2 Processor 60

4.2.3 Communications 62

4.2.4 Timers 63

4.3 The Structure of the Transputer Network 63

4.4 Connectivity of the Transputer System 65

4.4.1 Topology of transputer system 67

4.5 Algorithm Structure 69

4.5.1 Convolution 69

4.5.2 Backprojection 70

4.5.3 Interpolation 70

4.6 Implementation Details 71

4.6.1 Algorithm implementation in target system 71

4.6.2 Algorithm implementation in multiple transputer network system 72

Chapter 5 Conclusion 78

References 79

Appendix 82

List of Figures

Fig. 1-1 A digital image processing system 3

Fig. 1-2 Image representation and modelling 4

Fig. 1-3 Image reconstruction using x-ray CT scanners 7

Fig. 1-4 Typical chest x-ray radiograph 8

Fig. 1-5 The relationship between the projection of a two-dimensional function and slice of its Fourier transform 17

Fig. 2-1 Diagram to show that two summations can be performed in parallel 21

Fig. 2-2 Block diagram of the transputer 26

Fig. 2-3 Processes of a transputer 27

Fig. 3-1 Filter process 33

Fig. 3-2 Backprojection reconstruction..... 36

Fig. 3-3 The ideal filter response for the filtered backprojection algorithm 38

Fig. 3-4 The impulse response of the filter shown in Fig. 3-3 39

Fig. 3-5 Fan beam projections 41

Fig. 3-6 Fan and parallel beam projections 43

Fig. 3-7 Equi-Radial/Angular pixel geometry 47

Fig. 3-8 Block diagram of the hardware for a single block of the systolic array 49

Fig. 3-9 The blocks in systolic arrays 50

Fig. 3-10 The new block diagram of the hardware for a single block of the systolic array 51

Fig. 4-1 IMS T414 block diagram 58

Fig. 4-2 Linked process list 61

Fig. 4-3 The arrangement of transputer network 65

Fig. 4-4 Link names and addresses 66

Fig. 4-5 Block diagram of target system 68

Fig. 4-6 The communication structure of the transputer network 69

Fig. 4-7 Polar coodinate reconstruction geometry 74

Fig. 4-8 Cartesian reconstruction geometry 75

Fig. 4-9 Comparison of the reconstructed image with the original image .. 77

CHAPTER 1. INTRODUCTION TO X-RAY COMPUTERIZED TOMOGRAPHY

Tomography refers to the cross-sectional imaging of an object from either transmission or reflection data collected by illuminating the object from many different directions. The impact of this technique in diagnostic medicine has been revolutionary, since it has enabled doctors to view internal organs with unprecedented precision and with safety for the patient. The first medical application utilized x-rays for forming images of tissues based on their x-ray attenuation coefficient. More recently, however, medical imaging has also been successfully accomplished with radioisotopes, ultrasound, and magnetic resonance; the image parameter being different in each case.

There are numerous nonmedical imaging applications which lend themselves to the methods of computerized tomography. Researchers have already applied this methodology to the mapping of underground resources via crossborehole imaging, some specialized cases of cross-sectional imaging for nondestructive testing, the determination of the brightness distribution over a celestial sphere, and three-dimensional imaging with electron microscopy.

Fundamentally, tomographic imaging deals with reconstructing an image from its projections. It is an important part of Digital Image Processing. In this chapter, we firstly introduce some basic idea about Digital Image Processing. Then the fundamentals of computerized tomography will be discussed.

1.1 Digital Image Processing

Interest in digital image processing techniques dates back to the early 1920s when digitized pictures of world news events were first transmitted by submarine cable between New York and London. Applications of digital image processing concepts, however, did not become widespread until the middle 1960s, when third-generation digital computers began to offer the

speed and storage capabilities required for practical implementation of image processing algorithms. Since then, this area has experienced vigorous growth, having been a subject of interdisciplinary study and research in such fields as engineering, computer science, information science, statistics, physics, chemistry, biology, and medicine. The results of these efforts have established the value of image processing techniques in a variety of problems ranging from restoration and enhancement of space-probe pictures to processing of fingerprints for commercial transactions. Several new technological trends promise to further promote digital image processing. These include parallel processing made practical by low-cost microprocessors, and the use of charge-coupled devices (CCDs) for digitizing, Storage arrays. Another impetus for development in this field stems from some exciting new applications on the horizon. Certain types of medical diagnosis, including differential blood cell counts and chromosome analysis, are a state of practicality by digital techniques. The remote sensing programs are well suited for digital image processing techniques. Thus, with increasing availability of reasonably inexpensive hardware and some very important applications on the horizon, one can expect digital image processing to continue its phenomenal growth and to play an important role in the future.

1.1.1 The Elements of Digital Image Processing

Figure 1-1 shows a complete system for image processing. The digital image produced by the digitizer goes into temporary storage on a suitable device. In response to job control input, the computer calls up and executes image processing programs from a library. During execution, the input image is read into the computer line by line. Operating upon one or several lines, the computer generates the output data storage device line by line. During the processing, the pixels may be modified at the programmer's discretion in processing steps limited only by his imagination, patience, and computing budget. After processing, the final product is displayed by a process that is the reverse of digitization. The grey level of each pixel is used to determine the brightness (or darkness) of the corresponding point on a display screen. The processed image is thereby made visible and hence amenable to human interpretation.

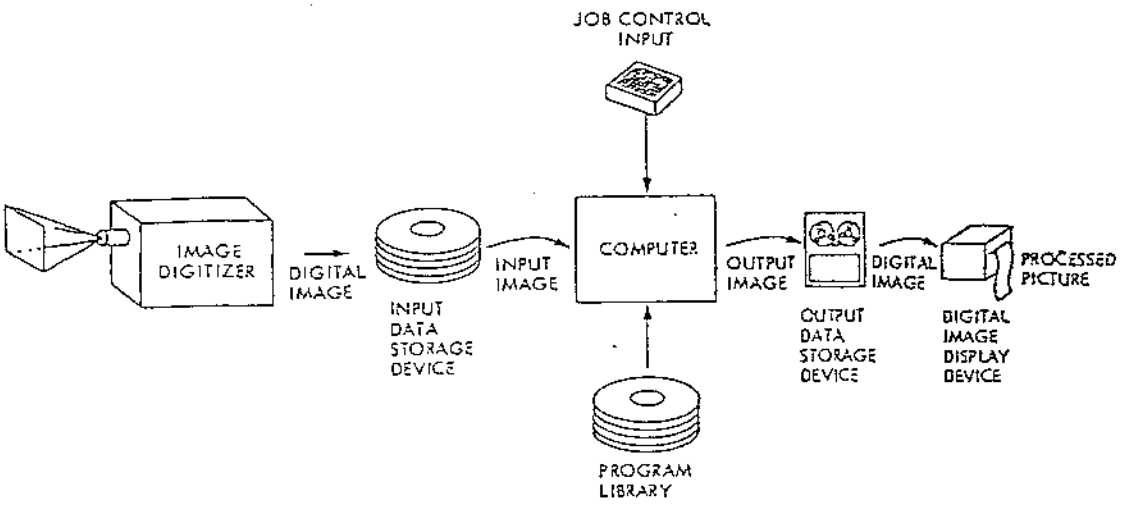


Fig. 1-1 A digital image processing system

1.1.2 Basic Classes of Digital Image Processing

Digital image processing has a broad spectrum of applications, such as remote sensing via satellites and other spacecraft, image transmission and storage for business applications, medical processing, radar, sonar, and acoustic image processing, robotics, and automated inspection of industrial parts.

Although there are many image processing applications, the basic classes of digital image processing are as follows:

* Image representation and modelling

In image representation one is concerned with characterization of the quantity that each picture-element (also called pixel) represents. An image could represent luminance of objects in a scene (such as pictures taken by ordinary camera), the absorption characteristics of the body tissue (X-ray imaging), the radar cross section of a target (radar imaging), the temperature profile of a region (infrared imaging). In general, any two-

dimensional function that bears information can be considered an image. Image models give a logical or quantitative description of the properties of this function. Figure 1-2 lists several image representation and modelling problems.

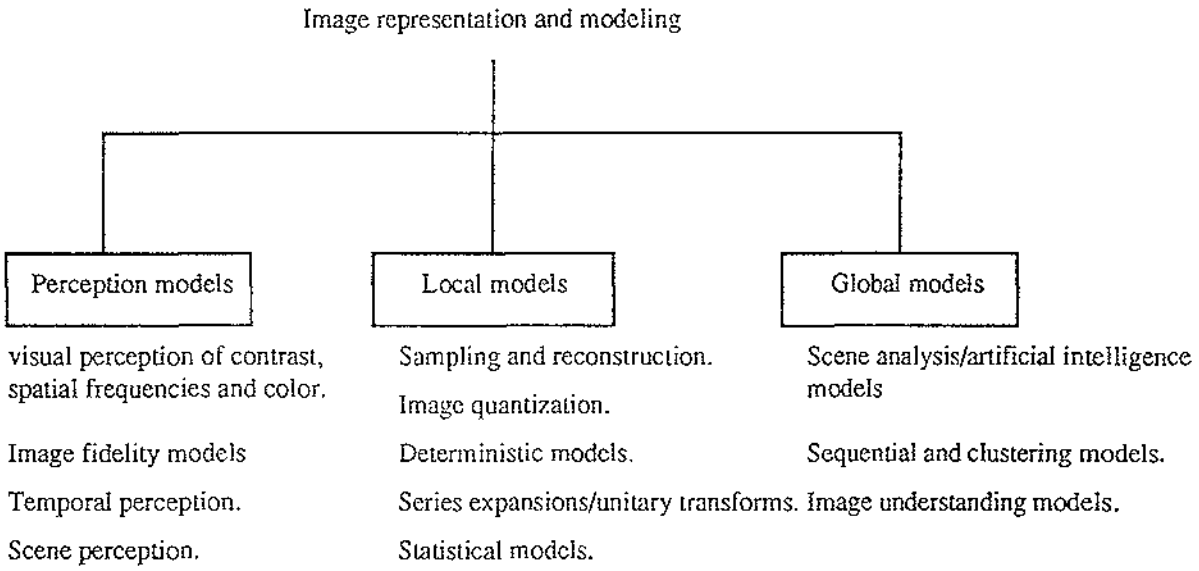


Fig. 1-2 Image representation and modelling

* Image enhancement

In *image enhancement*, the goal is to accentuate certain image features for subsequent analysis or for image display. Examples includes contrast and edge enhancement, pseudocoloring, noise filtering, sharpening, and magnifying. Image enhancement is useful in feature extraction, image analysis, and visual information display. The enhancement process itself does not increase the inherent information content in the data. It simply emphasises certain specified image characteristics. Enhancement algorithms are generally interactive and application-dependent.

* Image restoration

Image restoration refers to removal or minimisation of known degradations in an image. This includes deblurring of images degraded by the limitations

of a sensor or its environment, noise filtering, and correction of geometric distortion or nonlinearities due to sensors.

* Image analysis

Image analysis is concerned with making quantitative measurements from an image to produce a description of it. In the simplest form, this task could be reading a label on a grocery item, sorting different parts on an assembly line, or measuring the size and orientation of blood cells in a medical image. More advanced image analysis systems measure quantitative information and use it to make a sophisticated decision, such as controlling the arm of a robot to move an object after identifying it or navigating an aircraft with the aid of images acquired along its trajectory.

Image analysis techniques require extraction of certain features that aid in the identification of the object. Segmentation techniques are used to isolate the desired object from the scene so that measurements can be made on it subsequently. Quantitative measurements of object features allow classification and description of the image.

* Image data compression

The amount of data associated with visual information is so large (see Table 1.1a) that its storage would require enormous storage capacity. Although the capacities of several storage media (Table 1.1b) are substantial, their access speeds are usually inversely proportional to their capacity. Typical television images generate data rates exceeding 10 million bytes per second. There are other image sources that generate even higher data rates. Storage and/or bandwidth, which could be very expansive. Image data compression techniques are concerned with reduction of the number of bits required to store or transmit images without any appreciable loss of information. Image transmission applications are in broadcast television; remote sensing via satellite, aircraft, radar, or sonar; teleconferencing; computer communications; and facsimile transmission. Image storage is required most commonly for educational and business documents, medical images used in patient monitoring systems, and the like. Because of their wide

applications, data compression is of great importance in digital image processing.

Table 1.1a Data Volumes of Image Sources (in Millions of Bytes)

National archives	12.5×10^9
1 h of colour television	28×10^3
Encyclopaedia Britannica	12.5×10^3
Book (200 pages of text characters)	1.3
One page viewed as an image	0.13

Table 1.1b Storage Capacities (in Millions of Bytes)

Human brain	125,000,000
Magnetic cartridge	250,000
Optical disc memory	12,500
Magnetic disc	760
2400-ft magnetic tape	200
Floppy disc	1.25
Solid-state memory modules	0.25

* Image reconstruction from projections

Image reconstruction from projections is a special class of image restoration problems where a two (or higher) dimensional object is reconstructed from several one-dimensional projections. Each projection is obtained by projecting a parallel X ray (or other penetrating radiation) beam through the object (Figure 1-3). Planar projections are thus obtained by viewing the object from many different angles. Reconstruction algorithms derive an image of a thin axial slice of the object, giving an inside view otherwise unobtainable without performing extensive surgery. Such technique is referred to as computerized tomography; it has revolutionized diagnostic radiology over the past decade. The 1979 Nobel prize in medicine has been awarded for work on computerized tomography. The fundamentals of computerized tomography will be introduced in the next section.

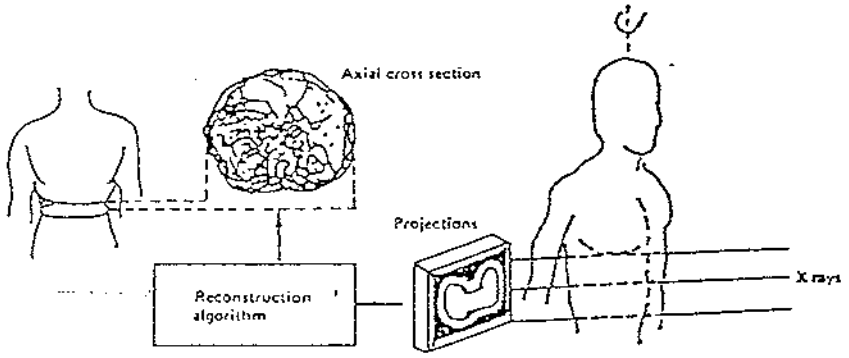


Fig. 1-3 Image reconstruction using x-ray CT scanners.

1.2. The Fundamentals of Computerized Tomography

We know that x- rays, radioisotopes, ultrasound and magnetic resonance can be used to obtain a reconstructed image. Because x-rays are broadly used today, we will only discuss utilizing x-rays to form the images based on their attenuation coefficient.

On November 1895, Professor Rontgen discovered the x-rays. The prospects for x-ray diagnosis were immediately recognised. In Great Britain, it has been estimated that there are 644 medical and dental radiography examinations per 1000 population per year, so that the technique is of major importance in medical imaging.

The radiographic image is formed by the interaction of x-ray photons with a photon detector and is therefore a distribution of those photons which are transmitted through the patient and are recorded by the detector. These photons can either be primary photons, which have passed through the patient without interacting or secondary photons, which result from an interaction in the patient. The secondary photons will in general be deflected from their original direction and carry little useful information. The primary photons do carry useful information. They give a measure of the probability that a photon will pass through the patient without interacting and this probability will itself depend upon the sum of the x-ray attenuating properties of all the tissues the photon traverses. The image is therefore a projection of the attenuating properties of all the tissues along the paths of the x-rays.

When we look at a chest x-ray (see Figure 1-4), certain anatomical features are immediately apparent. The ribs, for example, show up as a light structure because they attenuate the x-ray beam more strongly than the surrounding soft tissue, so the film receives less exposure in the shadow of the bone. Correspondingly, the air-filled lungs show up as darker regions.

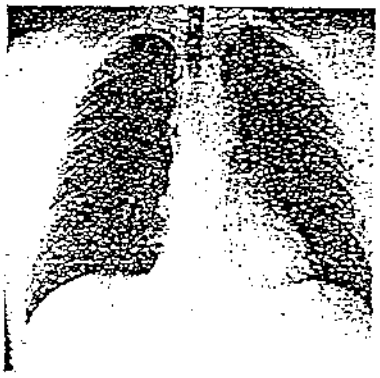


Fig. 1-4 Typical chest x-ray radiograph

X-ray films usually allow contrasts of the order of 2% to be seen easily, so a 1 cm thick rib or a 1 cm diameter air-filled trachea can be visualised. However, the blood in the blood vessels and other soft-tissue details, such as details of the heart anatomy, cannot be seen on a conventional radiograph. In order to make the blood vessels visible, the blood has to be infiltrated with a liquid contrast medium containing iodine compounds; the iodine temporarily increases the linear attenuation coefficient of the fluid medium to the point where visual contrast is generated. Consideration of photon scatter further degrades contrast.

Another problem with the conventional radiograph is the loss of depth information. The three-dimensional structure of the body has been collapsed, or projected, onto a two-dimensional film.

It is apparent that conventional x-radiographs are inadequate in these two respects, namely the inability to distinguish soft tissue and the inability to resolve spatially structures along the direction of x-ray propagation.

The announcement of a machine used to perform x-ray computerized tomography (CT) in a clinical environment, by Hounsfield at the 1972 British

Institute of Radiology annual conference, has been described as the greatest step forward in radiology since Rontgen's discovery. The relevant abstract (Ambrose and Hounsfield 1972) together with the announcement entitled 'X-ray diagnosis peers inside the brain' in the New Scientist (27 April 1972) can be regarded as the foundation of clinical x-ray CT. Hounsfield shared the 1979 Nobel Prize for Physiology and Medicine with Cormack.

With computerized tomography, the two inabilities of conventional x-radiographs can be solved. By combining "ordinary" x-ray technology with sophisticated computer signal processing, computerized tomography can generate a display of the tissues of the body which is unencumbered by the shadows of other organs. Computerized tomography also passes x-rays through the body of a patient, but the detection method is usually electronic in nature, and the data is then converted from an analogue signal to digital impulses in an analogue-to-digital (A/D) converter. This digital representation of the x-ray intensity is fed into a computer, which then reconstructs an image.

Since Hounsfield's invention of the computerized tomography (CT) scanner in 1972, great improvements have been made in x-ray tomography, with the result that the patient scan time has been reduced to less than 10s from over 4 mins.

1.2.1. Theoretical Background for Image Reconstruction

1.2.1.1. The N-Dimensional Continuous Fourier Transform (CFT)

We consider here a function $f(x_1,x_2,...,x_N)$ of N continuous variables $x_1,x_2,...,x_N$. We will generally find it convenient to express the N-tuple $(x_1,x_2,...,x_N)$ as a vector $\vec{x}$ and refer to the function as $f(\vec{x})$. The N-dimensional Fourier transform of $f(\vec{x})$ is denoted by $F(\omega_1,\omega_2,...,\omega_N)$ or $F(\vec{\omega})$. The domain of f(x) will be referred to as signal space and the domain of $F(\vec{\omega})$ as Fourier space. The N-dimensional function $f(\vec{x})$ and its Fourier transform are related by:

$$F(\omega_1,\omega_2,\ldots,\omega_N)=\int_{-\infty}^{+\infty}\ldots\int_{-\infty}^{+\infty}f(x_1,x_2,\ldots,x_N)\exp[-j(\omega_1x_1+\omega_2x_2+\ldots+\omega_Nx_N)]dx_1dx_2\ldots dx_N \tag{1.1}$$

$$f(x_1,x_2,\ldots,x_N)=\frac{1}{(2\pi)^N}\int_{-\infty}^{+\infty}\ldots\int_{-\infty}^{+\infty}F(\omega_1,\omega_2,\ldots,\omega_N)\exp[j(\omega_1x_1+\omega_2x_2+\ldots+\omega_Nx_N)]d\omega_1d\omega_2\ldots d\omega_N \tag{1.2}$$

or, expressed in vector notation,

$$F(\vec{\omega})=\int_{-\infty}^{+\infty}f(\vec{x})\exp[-j(\vec{x}\cdot\vec{\omega})]d\vec{x} \tag{1.3}$$

and

$$f(\vec{x})=\frac{1}{(2\pi)^N}\int_{-\infty}^{+\infty}F(\vec{\omega})\exp[j(\vec{x}\cdot\vec{\omega})]d\vec{\omega} \tag{1.4}$$

where $\vec{x}\cdot\vec{\omega}$ denotes the dot product of the vectors $\vec{x}$ and $\vec{\omega}$ or equivalently with $\vec{x}$ and $\vec{\omega}$ interpreted as row matrices $\vec{x}\cdot\vec{\omega}=\vec{x}\vec{\omega}'$.

A useful property of the N-dimensional Fourier transform pair which we will want to use later is that if $f(\vec{x})$ and $F(\vec{\omega})$ form a Fourier transform pair, then $f(\vec{x}\vec{A})$ and $F(\vec{\omega}\vec{A})$ form a Fourier transform pair if A is an orthogonal matrix, i.e., $\vec{A}'=\vec{A}^{-1}$.

This property is easily verified by direct substitution into (1.3). Thus an orthogonal transformation or equivalently an orthogonal change of coordinates in signal space results in the same change of coordinates in Fourier space. For example, for N=2, if

$$A=\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix} \tag{1.5}$$

so that $f(\vec{x})$ is rotated by an angle θ, then its Fourier transform will be rotated by the same angle θ.

1.2.1.2. The N-Dimensional Discrete Fourier Transform (DFT)

We are here interested in functions which can be processed by a digital computer and consequently can be represented by their samples. Thus we consider the class of band-limited functions. Specifically, $f(\vec{x})$ is said to be band-limited if there exists an N-tuple $(W_1, W_2, ..., W_N)$ such that $F(\vec{\omega})$ is zero for $|\omega_i| > W_i$, $i = 1,2,...,N$. In some cases it is convenient to set $W = \max(W_1, W_2, ..., W_N)$ and refer to the scalar W as the bandwidth of $f(\vec{x})$.

The N-dimensional sampling theorem states that if $f(\vec{x})$ is sampled in signal space on a rectangular lattice with the sample spacing in dimension x_i less than π / W_i, then $f(\vec{x})$ can be recovered from its samples. Sampling on a rectangular lattice will be referred to as periodic Cartesian sampling.

Let us denote by $g(\vec{n})$ the N-dimensional sequence corresponding to sampling $f(\vec{x})$ with a sample spacing of π / V_i in the dimension x_i where $V_i > W_i$ so that

$$g(n_1, n_2, ..., n_N) = f\{\frac{\pi n_1}{V_1}, \frac{\pi n_2}{V_2}, ..., \frac{\pi n_N}{V_N}\} \tag{1.6}$$

from the sampling theorem the Fourier transform $F(\vec{\omega})$ of $f(\vec{x})$ can be obtained from $g(\vec{n})$ by the relation

$$F(\vec{\omega}) = \frac{\pi^N}{V_1 V_2 \cdots V_N} \{\sum_{-\infty}^{+\infty} g(\vec{n}) \exp[-j\pi(\vec{n} \cdot \vec{\omega}_V)]\} \cdot b_V(\omega) \tag{1.7}$$

where $\vec{\omega}_V$ denotes the vector $(\frac{\omega_1}{V_1}, \frac{\omega_2}{V_2}, ..., \frac{\omega_N}{V_N})$ and

$$b_V(\omega) = \begin{cases} 1, & |\omega_i| < V_i, \quad i = 1,2,...,N \\ 0, & \text{otherwise} \end{cases}$$

likewise, the sequence $g(\vec{n})$ can be obtained from $F(\vec{\omega})$ by the relation

$$g(\vec{n}) = \frac{1}{(2\pi)^N} \int_{-V}^{+V} F(\vec{\omega}) \exp\{j\pi(\vec{n} \cdot \vec{\omega}_V)\} d\vec{\omega} \tag{1.8}$$

The original N-dimensional function $f(\vec{x})$ can be obtained from the sequence $g(\vec{n})$ by means of the interpolation formula

$$f(\vec{x}) = \sum_{n=-\infty}^{+\infty} g(\vec{n}) \phi(\vec{n}, \vec{x}) \tag{1.9}$$

where

$$\phi(\vec{n}, \vec{x}) = \prod_{i=1}^{N} \frac{\sin V_i (x_i - \frac{n_i \pi}{V_i})}{V_i (x_i - \frac{n_i \pi}{V_i})} \tag{1.10}$$

When only a finite number of the samples of $f(\vec{x})$ are nonzero, the Fourier transform $F(\vec{\omega})$ can be represented by a finite set of Cartesian samples. The relationship between the Cartesian samples of $F(\vec{\omega})$ and the Cartesian samples of $f(\vec{x})$ is the N-dimensional DFT. Specifically, let us assume that

$$g(\vec{n}) = 0, \quad \text{if } n_i \geq M_i \text{ or } n_i < 0, i = 1,2,...,N.$$

We now consider the Cartesian samples of $F(\vec{\omega})$, which we denote by $G(\vec{k})$ given by

$$G(k_1, k_2, ..., k_N) = \frac{\pi^N}{V_1 V_2 ... V_N} F(\frac{2V_1}{M_1} k_1, \frac{2V_2}{M_2} k_2, ..., \frac{2V_N}{M_N} k_N)$$

where k_i is an integer such that

$$-\frac{M_i}{2} + 1 \leq k_i \leq \frac{M_i}{2}, \text{ if } M_i \text{ is even}$$

$$-\frac{M_i - 1}{2} \leq k_i \leq \frac{M_i - 1}{2}, \text{ if } M_i \text{ is odd}$$

Then

$$G(k_1,k_2,...,k_N)=\sum_{n_1=0}^{M_1-1}...\sum_{n_N=0}^{M_N-1}g(n_1,n_2,...,n_N)\cdot\exp[-j2\pi(\frac{k_1n_1}{M_1}+\frac{k_2n_2}{M_2}+...+\frac{k_Nn_N}{M_N})] \tag{1.11}$$

and

$$g(n_1,n_2,...,n_N)=\frac{1}{M_1\cdot M_2\cdot...\cdot M_N}\sum_{k_1}...\sum_{k_2}G(k_1,k_2,...,k_N)\cdot\exp[j2\pi(\frac{k_1n_1}{M_1}+\frac{k_2n_2}{M_2}+...+\frac{k_Nn_N}{M_N})] \tag{1.12}$$

Since $G(k_1,k_2,...,k_N)$ as defined in (1.11) is periodic in k_i it is frequently convenient to use the values of k_i in the range given above. Adopting this convention and defining the vector $\vec{k}_M$ as the N-tuple

$$(\frac{k_1}{M_1},\frac{k_2}{M_2},...,\frac{k_N}{M_N})$$

we can express (1.11) and (1.12) as

$$G(\vec{k})=\sum_{n=0}^{M-1}g(\vec{n})\cdot\exp[-j2\pi(\vec{n}\cdot\vec{k}_M)] \tag{1.13}$$

and

$$g(\vec{n})=\frac{1}{M_1M_2...M_N}\sum_{k=0}^{M-1}G(\vec{k})\exp[j2\pi(\vec{n}\cdot\vec{k}_M)] \tag{1.14}$$

Equation (1.13) and (1.14) are referred to as the N-dimensional DFT pair. The N-dimensional DFT can be computed efficiently by using the one-dimensional fast Fourier transform (FFT) algorithm, since the summations in (1.13) and (1.14) can each be decomposed as a cascade of one-dimensional transforms.

The class of functions $f(\vec{x})$ which can be represented by a finite number of samples will be referred to as band-limited functions of finite order M where

$$M = \max\{M_1, M_2, \ldots, M_N\}.$$

1.2.1.3. Projections

A projection is a mapping of an N-dimensional function to an (N-1)-dimensional function obtained by integrating the function in a particular direction. For example, $P_{x_2}(x_1)$ given by

$$P_{x_2}(x_1) = \int_{-\infty}^{+\infty} f(x_1, x_2) dx_2 \tag{1.15}$$

is an example of a projection of the two-dimensional function $f(x_1, x_2)$ onto one dimension.

For the general case, we define a projection as follows: Let $f(\vec{x})$ denote an N-dimensional function and let $\vec{u}$ denote a new set of coordinates where

$$\vec{x} = \vec{u}\mathrm{A}$$

and A is an orthogonal transformation. Then a projection onto the hyperplane $(u_1, u_2, \ldots, u_{i-1}, u_{i+1}, \ldots, u_N)$ is defined as

$$p_{u_i}(u_1, u_2, \ldots, u_{i-1}, u_{i+1}, \ldots, u_N) = \int_{-\infty}^{+\infty} f(\vec{u}\mathrm{A}) du_i \tag{1.16}$$

The coordinate axis u_i, which is normal to the hyperplane onto which $f(\vec{x})$ is projected, will be referred to as the projection axis.

For N=2, the matrix A is given by

$$\mathrm{A} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

In this case, the u_1, u_2 coordinate axes are offset from the (x_1,x_2) axes by an angle of θ. For two-dimensional functions it will generally be convenient to refer to a projection by its angle θ. A projection at angle θ will be interpreted to mean a projection onto the coordinate u_1, which is at an angle θ with x_1. Equivalently, then, the projection axis u_2 is at an angle θ to the coordinate axis x_2. Equation (1.15) corresponds to a projection at an angle $\theta = 0$ or equivalently with x_2 as the projection axis.

1.2.1.4. The Projection-Slice Theorem

The projection-slice theorem relates the (N-1)-dimensional Fourier transforms of the projections to the N-dimensional Fourier transform of the original function. Basically, the theorem states that the (N-1)-dimensional Fourier transform of a projection is a "slice" through the N-dimensional Fourier transform of $f(\vec{x})$.

First, let us consider a projection for which the projection axis is one of the coordinate axes of $f(\vec{x})$, for example, x_1. Then $p_{x_1}(x_2,...,x_N)$ is given by

$$p_{x_1}(x_2,...,x_N) = \int_{-\infty}^{+\infty} f(\vec{x})dx_1 \tag{1.17}$$

and its (N-1)-dimensional Fourier transform is given by

$$p_{x_1}(\omega_2,...,\omega_N) = \int_{-\infty}^{+\infty} ... \int_{-\infty}^{+\infty} p_{x_1}(x_2,...,x_N) \cdot \exp[-j(\omega_2 x_2 + ... + \omega_N x_N)] \tag{1.18}$$

Comparing (1.18) and (1.1), we see that

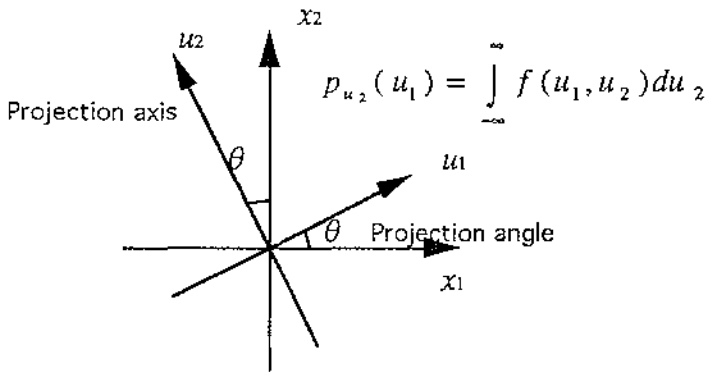
$$p_{x_1}(\omega_2,...,\omega_N) = F(\omega_1,\omega_2,...,\omega_N)|_{\omega_1=0} \tag{1.19}$$

In other words, $p_{x_1}(\omega_2,...,\omega_N)$ is a "slice" of $F(\omega_1,\omega_2,...,\omega_N)$ defined by $\omega_1 = 0$. Clearly, a projection whose axis is any coordinate axis x_i has a Fourier transform that is a slice of $F(\omega_1,\omega_2,...,\omega_N)$ defined by $\omega_i = 0$.

A general projection was defined in (1.16) where A is an orthogonal transformation. It was argued previously that if $F(\vec{\omega})$ is the Fourier transform of f($\vec{x}$) then $F(\vec{\Omega})$ is the Fourier transform of $f(\vec{u})$ where

$$\vec{x} = \vec{u}A$$
$$\vec{\omega} = \vec{\Omega}A \qquad (1.20)$$

From this, (19) is easily generalized to state that a projection for which the projection axis is the transformed coordinate u_i has an (N-1)-dimensional Fourier transform which is a slice of $F(\vec{\Omega})$ for $\Omega_i = 0$ where the coordinate systems $\vec{u}$ and $\vec{\Omega}$ are related to the coordinate systems $\vec{x}$ and $\vec{\omega}$ by the same orthogonal transformation. In two dimensions, for example, the projection-slice theorem states that the one-dimensional Fourier transform of a projection at an angle θ is a slice at the same angle of the two-dimensional Fourier transform of the original object. This relationship is depicted in Figure 1-5.



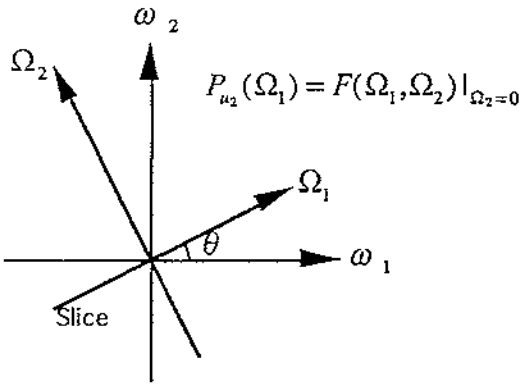


Fig. 1-5 The relationship between the projection of a two-dimensional function and slice of its Fourier transform.

1.2.1.5. The Basis for Reconstruction from Projections

From the projection-slice theorem, we see that specification of a projection in signal space corresponds to the specification of a slice in Fourier space and thus represents a partial specification of the signal itself. In principle, then, if an unlimited number of projections at different orientations are available, the Fourier transform of $f(\vec{x})$ can be obtained and therefore so can $f(\vec{x})$ itself. Generally, in any practical context, we are restricted to a finite number of projections.

Under certain assumptions, it is possible to carry out an exact reconstruction from a finite number of projections. If the structure is highly symmetrical, a finite number of projections might suffice for exact reconstruction. For example, for a two-dimensional circularly symmetric function all of its projections are identical and consequently such a function can be represented exactly by a single projection. Similarly, in three dimensions, for an object which is cylindrically symmetric all of the projections for which the axis is normal to the longitudinal axis are identical and consequently, in this case also, a single projection is sufficient.

In uitilizing projections for reconstruction, many of the algorithms involve computing the Fourier transforms of the projections. The Fourier transform

of each projection is a function of a set of continuous variables, but only a finite number of points from each Fourier transform can be computed and stored. Thus from the projections, only samples in the Fourier domain are available, in part because of the limited number of projections and in part because only samples of the Fourier transform on each slice can be obtained. The essence of the reconstruction problem, then, is to approximate all of Fourier space from its values on a discrete point set.

1.2.1.6. Reduction of the Dimensionality of the Reconstruction Problem

As we saw in Section 4, the underlying basis for reconstruction is to obtain samples in the Fourier plane by transforming projections. Intuitively it seems reasonable that projections need not be taken in all orientations. For three-dimensional objects, for example, we could imagine using only projections in the spatial domain on planes parallel to one of the coordinates axis, say x_1. Slices of these projections at $x_1 = A$ are then projections of the two-dimensional function $f(A,x_2,x_3)$ and, consequently, a two-dimensional reconstruction algorithm can be applied to reconstructing this two-dimensional slice of the three-dimensional object. In this way, the three-dimensional object can be built up slice by slice and, consequently, the three-dimensional problem can be reduced to a series of two-dimensional problems. In the general case, we can apply a similar argument to reduce an N-dimensional problem to a set of (N-1)-dimensional problems each of which can in principle be reduced to an (N-2)-dimensional problem, etc.

Thus in principle, an N-dimensional problem can be reduced to a set of two-dimensional problems. Often this procedure requires considerably less storage and is simpler computationally than solving the N-dimensional problem directly. Furthermore, in many cases, we may be content with very coarse sampling in one or several dimensions. For the three-dimensional problem, for example, reconstruction of only a few slices may be sufficient.

CHAPTER 2. INTRODUCTION TO PARALLEL PROCESSING AND TRANSPUTERS

2.1. The Principles of Parallel Processing Systems

In real life, things occur in parallel. Consider how unrealistic it would be if that were not the case. It would be most strange if, whilst waiting for a roll of film to be developed at the printer, the photographer did not continue to take photographs, assuming both film and camera were available and operative.

The manager who needs to present information in a meaningful way, does not wait until a report has been typed before undertaking any further work. To be efficient, these tasks must be carried out in parallel.

The reasons for finding the parallel method of operation preferable to the sequential mode are quite apparent. The sequential mode leads to a very uneconomical time management function, being a tremendous waste of time and resources.

It is the same that a simple-minded approach to gain speed, as well as power, in computing is through parallelism; here many computers would work together, all simultaneously executing some portions of a procedure used for solving a problem. Such an approach rests on the following assumptions:

(1) The availability of many low-cost, high speed computers that can be put together to work in unison, as in a concert;

(2) the existence of a strategy to partition a problem into smaller problems, such that most of these can be solved simultaneously, and from which we can easily construct the solution to the entire problem: this is popularly known as the "divide-and-conquer strategy".

Fortunately, the first of these assumptions is facilitated by the recent advances in microelectronics, solid state and superconduction devices technology. Using very large scale integration (VLSI) it is now possible to

fabricate millions of transistor-equivalent devices on a single 4mm square silicon ship. Several hundreds of thousands of such chips can be put together to build several thousands of processors within a few cubic centimetres at a reasonable cost.

The second of these assumptions, namely the application of the divide-and-conquer strategy, raises three basic issues:

(1) decomposability,
(2) complexity, and
(3) communication.

We shall consider these issues below.

Decomposability.

Decomposibility is a measure of the degree by which a problem can be split into components which can be computed in parallel. Some problems appear to be inherently serial in nature and not amenable to parallel processing. It is however not easy to determine whether a problem is inherently serial as it is often possible to reformulate problems into a form suitable for parallel processing.

A very simple example is the sum of a set of figures. This appears to be an inherently serial problem. However if we reformulate the problem in a hierachical form, the addition of pairs of numbers may be partially computed in parallel.

e.g.. A=1,2,5,7,9

Let S=sum(A) and formulate the solution as in figure 2-1, then two summation may be performed in parallel.

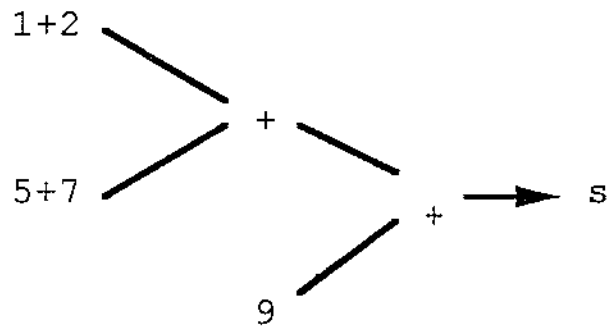


Fig. 2-1. Diagram to show that two summation can be performed in parallel

Complexity.

As mentioned before, to obtain the maximum gain in speed, several processors should be used to process the decomposed tasks. The effectiveness of this depends on how well a problem can be partitioned for solution by a given computer architecture or by an algorithm.

For instance, let T be the total time taken by an algorithm in which there is a serial portion taking time T_s and a parallel portion taking time T_p, so that $T_s + T_p = T$; then, no matter how many processors are used, the serial portion would limit the increase in speed by a factor of at most T / T_s; this is because at best T_p can go down to near zero! Thus, if an algorithm has a 10% serial portion, the speed increase that can be achieved by putting infinitely many processors to work would still be limited to ten times.

Therefore a judicious choice is needed in partitioning an algorithm and minimizing the serial portion.

Communication.

When a large problem is broken into smaller tasks, it is necessary to set up coordination between these tasks. To obtain this coordination we need communication links between the different tasks. The larger the number of pieces a problem is split into, the more communication links the resulting algorithm requires. In fact, the number of directed two-way communication

links between any two tasks among n tasks could be n(n-1) and so the communication complexity grows quadratically. The communication problem introduces a new dimension to parallel programming and parallel architecture design. This means that in addition to the computing time and memory space requirements of an algorithm, we must also consider the communication costs and set a bound on the complexity of communication. This would imply that communication between tasks which are not close enough (closeness being measured by a suitably defined criterion for an architecture) should be avoided; that is, the communication should preferably be confined to only those processes that are very close neighbours.

When a large number of processors are assigned to carry out the split tasks, it is possible that simultaneous request or access to certain data or tasks may create a conflict or collision. This could slow down the anticipated speed advantage resulting from the multiple processors, or may even lead to a state of inactivity or standstill when two processes or processors wait for each other indefinitely (deadlock), or may delay some process indefinitely (lockout). Since the computational speeds for different tasks are unpredictable (non-deterministic), the different processes may loss synchronisation, leading to a total breakdown of the tasks.

The communication problem is therefore concerned with the minimisation of communication complexity, prevention of deadlocks and improved coordination.

In recent years, the models and techniques employed to solve the three basic issues, namely decomposability, complexity and communication, have grown into a major interdisciplinary science of parallel processing. This science deals with both the theoretical studies and the practical aspects of design to achieve the best results.

2.2. Description of Parallel Processes

Traditionally, computer users are taught to program processes which are essentially sequential. The sequential nature gives rise to a deterministic set

of operations. Therefore, a sequential program is easily described with the help of flow diagrams containing the following well-known deterministic constructs (or boxes):

(1) begin (start);
(2) assignment (set);
(3) if B then S else T (decision);
(4) while B do S (indefinite loop);
(5) for i=1 to n (definite loop);
(6) end (halt).

In the case of parallel or non-sequential programs, in addition to the above constructs we need the following mechanisms:

(1) parallel initiation and termination mechanisms;
(2) synchronisation, protection and communication mechanisms;

2.2.1. Initiation and Termination Mechanisms

When a set of processes are non-sequential we need to have constructs that can begin a list of processes simultaneously and end a list of processes when all of them terminate. These are respectively called cobegin and coend constructs.

2.2.2. Synchronization Mechanism

When a set of processes are non-sequential, the processes may have several simultaneous inputs or outputs and they may need to pass data to one another or to communicate. Since the speed at which each process works is not determinable, non-determinism arises in linking the processes. For instance, even if several processes begin together they may not all end simultaneously. Therefore we say there is a *race condition* and the parts of computation containing such processes are *time critical*. A need therefore arises to introduce a mechanism to coordinate and control the temporal order in which processes are executed to realise a given non-sequential algorithm. Such a mechanism is called a *synchronisation mechanism*.

2.2.3. Protection Mechanism

The synchronisation mechanism alone is not adequate to carry out non-sequential programming. In order to coordinate the processes it is also necessary to prevent clashes among them. That is, we must restrict the access of a shared variable or a resource to one process at a time. This is achieved by using another mechanism called the protection mechanism. This protection mechanism prevents interference or clashes from other processes when a particular process is using a shared variable or a resource.

The sequence of statements in which access to a shared variable or resource is to be exclusively provided to a process is called a critical section or critical region. When a process is about to execute its critical section we avoid a clash by ensuring that no other process is executing its own critical region at the same time. Then once access is given to a process, access by another process should be allowed by an unlocking process. In other words, the protection mechanism should provide a locking facility to allow each process to perform its critical section and to unlock and let the other processes do their own critical sections.

2.2.4. Communication Mechanism

Two distinctly different methods are used for synchronisation among processes.

(1) Shared-variable method.

In this method, synchronisation and communication among the processes are achieved using shared mechanisms under a centralised control.

(2) Message-passing method.

In this method the processes are autonomously controlled by sending and receiving messages, without sharing data; they are coordinated by using

delay and wait operations among them. These two methods have contributed to the development of new styles of programming languages for concurrent programming.

The design of methods for the description of parallel processes and the synchronisation and protection mechanisms has been a very active area of research during recent years. As a result of these studies, new concepts and analytic models have been developed for the programming, complexity and analysis of parallel and concurrent processes.

2.3. The Transputer

The transputer comprises a single chip, made using very large scale integration (VLSI) techniques which condense the equivalent of 250,000 transistors onto a chip measuring 9mm square. Some of the transputer's circuitry and elements are as small as 1.5 micrometers.

The objective of the transputer is to generate high performance calculations and currently, the most powerful transputer can process 10 million program instructions per second. This is faster than any other 32-bit microprocessor and has the advantage of being increased when working in conjunction with other transputers.

The transputer does not operate using a series of sequential steps, but is a parallel processor with the ability to carry out several computations simultaneously. The management of this parallel facility may be carried out via the purpose-built concurrent programming language Occam.

2.3.1. Overview

Typically, the transputer comprises a single chip with processor, 2K or 4K of on-chip memory and communications links. Additional special circuitry needed to adapt it to a particular use, e.g. a disk controller, may also be included.

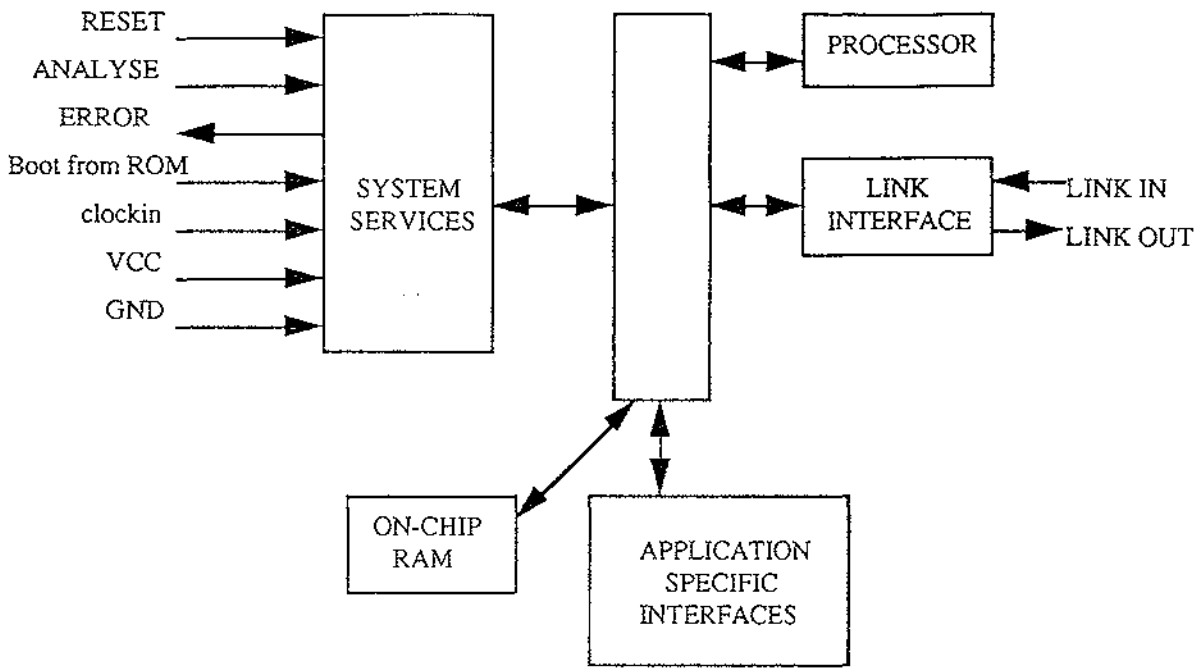


Fig. 2-2 Block diagram of the transputer

The transputer has been specifically developed for concurrent processing. The on-chip local memory assists in eliminating processor-to-memory bottlenecks, and each transputer supports 4 asynchronous, high speed serial links to other transputer units. The efficient utilisation of each processor's time slices is carried out by a microcoded scheduler.

The transputer-to-transputer links provide a combined data communications capacity of 5 Megabytes/sec and operate concurrently with internal processes. This is a radical departure from the shared bus concept employed in the majority of multiprocessor architectures. It allows parallel connection without the overhead resulting from having to provide the complex communications between conventional parallel processors. The advantages over multi-processor buses are as follows:

- No contention for communications
- No capacity load penalty as transputers are added
- The bandwidth does not become saturated as the system increases in size.

The system architecture may comprise either a single transputer chip or a network of transputers functioning together as a concurrent system.

Low cost systems which perform at supercomputer speeds are quite easily achieved. This is because when transputers are connected together there is an increase in processing power available which, as there is no communication overhead, theoretically has no limit.

2.3.2. Occam

The system supports a high level concurrent programming language, Occam, specifically designed to run efficiently on transputer systems. Occam allows access to machine features and removes the need for a low level assembly language.

Occam and the transputer were designed together enabling optimal implementations of Occam to be achieved with respect to concurrency and communications. The transputer instruction sets have been designed for efficient and simple Occam compilation.

An Occam program comprises a set of sequential or parallel processes, or may comprise a number of other processes.

There are three primitive processes upon which all else is based, these being assignment , input and output.

Fig. 2-3. Processes of a transputer

An assignment changes the value of a variable, an input receives a value from a channel, and an output sends a value to a channel.

In Occam, an assignment is indicated by the symbol :=. The example

```
v := e
```

sets the value of the variable **v** to the value of the expression **e** and then terminates.

An input is indicated by the symbol **?** The example

```
c ? x
```

inputs a value from channel **c**, assigns it to the variable **x** and then terminates.

An output is indicated by the symbol **!** The example

```
c ! e
```

outputs the value of the expression **e** to channel **c**.

Sequential programming languages deal with processes as a number of statements to be executed strictly in sequence. In the real world, just as a large, work intensive task would be split between many people in order to utilise the available workforce and to improve performance levels, so a process can be expressed in terms of modules which may be executed concurrently. The Occam language enables an application to be expressed in terms of a number of concurrent processes which communicate using channels.

For example, Occam can express an application as follows:

```
PAR
  P1
  P2
  P3
  ...
```

The component processes P1, P2, P3 ... are executed together, and are called concurrent processes. The construct terminates after all of the component processes have terminated, for example:

```
PAR
  c1 ? x
  c2 ! y
```

allows the communications on channels c1 and c2 to take place together.

The parallel construct is unique to OCCAM. It provides a straightforward way of writing programs which directly reflects the concurrency inherent in real systems.

Occam can be applied to a system with one transputer or one which is made up of many transputers. A program written in Occam may be implemented on a single transputer, or, without modification, (although some modification may enhance performance) it may be implemented on a system with more than one transputer. In this situation each transputer would be executing an Occam process, and the links between the transputers would provide the Occam channels. Occam may be used as a design language for such a system, describing both the system as a whole, as well as each individual transputer.

OCCAM programs may be configured for execution on one or many transputers. The transputer development system provides the necessary tools for correctly distributing a program configured for many transputers.

Configuration does not affect the logical behaviour of a program. However, it does enable the program to be arranged to ensure that performance requirements are met.

PLACED PAR

A parallel construct may be configured for a network of transputers by using the PLACED PAR construct. Each component process (termed a placement)

is executed by a separate transputer. The variables and times used in a placement must be declared within each placement process.

PRI PAR

On any individual transputer, the outermost parallel construct may be configured to prioritise its components. Each process is executed at a separate priority. The first process has the highest priority, the last process has the lowest priority. Lower priority components may only proceed when all higher priority components are unable to proceed.

2.3.3. Communication Channels

Two processes or equivalently, two transputers, communicate via channels. These channels are one-way with one process writing to the channel and the other process reading from the channel. When it is required that communications between processes takes place, the two processes must synchronise, that is, they must both be ready. If necessary, one process must wait for the other.

Two processes communicating is equivalent to two transputers communicating; this mode of communication is similar to the handshake method used in hardware systems.

Internally, there may be many concurrently performing communications channels because Occam may be used for a single transputer or a whole network of maybe thousands. If the system comprises a single transputer then the processor time is shared between the concurrent processes with channels being achieved by a block movement of data in memory.

If a network of transputers makes up the system then each may execute an individual process and communications are carried out directly via explicit transputer-to-transputer links. The same machine instruction is used whether a single transputer or many are being utilised.

Concurrent processes communication only by using channels, and communication is synchronized. If a channel is used for input in one process, and output in another, communication takes place when both the inputting and the outputting processes are ready. The value to be output is copied from the outputting process to the inputting process, and the processes then proceed.

Communication between processes on a single transputer is via memory-to-memory data transfer. Between processes on different transputers it is via standard links. In either case the OCCAM program is identical.

CHAPTER 3. RECONSTRUCTION ALGORITHMS FOR PARALLEL AND FAN BEAM PROJECTIONS

The projection slice theorem [5] relates the Fourier transform of a projection to the Fourier transform of the object along a single radial. Thus given the Fourier transform of a projection at enough angles the projections could be assembled into a complete estimate of the two-dimensional transform and then simply inverted to arrive at an estimate of the object, while this provides a simple conceptual model of tomography, practical implementations require a different approach.

The algorithm that is currently being used in almost all applications of straight ray tomography is the filtered backprojection algorithm. It has been shown to produce accurate reconstructions and amenable to fast implementation, it is derived from the Projection Slice Theorem. This theorem is brought into play by rewriting the inverse Fourier transform in polar coordinates and rearranging the limits of the integration therein. The derivation of this algorithm is perhaps one of the most illustrative examples of how we can obtain a radically different computer implementation by simply rewriting the fundamental expressions for the underlying theory.

In this chapter, details will be presented for the filtered backprojection algorithms for two types of scanning geometries, parallel beam and fan beam. Then, a modified reconstruction algorithm will be advanced, it reduces the computational requirements of backprojection. The computer implementation of these algorithms requires the projection data to be sampled and then filtered. We will first provide an intuitive rationale behind the filtered backprojection approach.

3.1 Reconstruction Algorithm for Parallel Projections

3.1.1. The Idea

The filtered backprojection algorithm can be given a rather straightforward intuitive rationale because each projection represents a nearly independent measurement of the object. This isn't obvious in the space domain but if the

Fourier transform is found of the projection at each angle then it follows easily by the Projection Slice Theorem [5]. We say that the projections are nearly independent (in a loose intuitive sense) because the only common information in the Fourier transforms of the two projections at different angles is the dc term.

To develop the idea behind the filtered backprojection algorithm, we note that because of the Fourier Slice Theorem the act of measuring a projection can be seen as performing a two-dimensional filtering operation. Consider a single projection and its Fourier transform. By the Fourier Slice Theorem, this projection gives the values of the object's two-dimensional Fourier transform along a single line. If the values of the Fourier transform of this projection are inserted into their proper place in the object's two-dimensional Fourier domain then a simple (albeit very distorted) reconstruction can be formed by assuming the other projections to be zero and finding the two-dimensional inverse Fourier transform. The point of this exercise is to show that the reconstruction so formed is equivalent to the original object's Fourier transform multiplied by the simple filter shown in Figure 3-1.

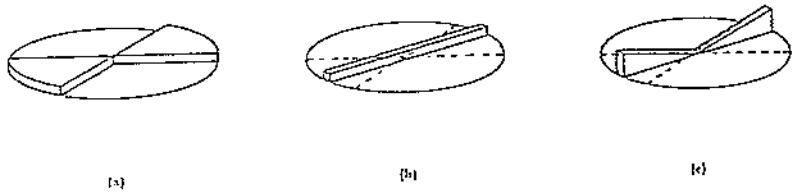


Fig. 3-1 Filter process

This figure shows the frequency domain data available from one projection. (a) is the ideal situation. A reconstruction could be formed by simply summing the reconstruction from each angle until the entire frequency domain is filled. What is actually measured is shown in (b). As predicted by the Fourier Slice Theorem, a projection gives information about the Fourier transform of the object along a single line. The filtered backprojection algorithm takes the data in (b) and applies a weighting in the frequency domain so that the data in (c) are an approximation to those in (a).

What we really want from a simple reconstruction procedure is the sum of projections of the object filtered by pie-shaped wedges as shown in Figure 3-1.

It is important to remember that this summation can be done in either the Fourier domain or in the space domain because of the Fourier transform. As will be seen later, when the summation is carried out in the space domain, this constitutes the backprojection process.

As the name implies, there are two steps to the filtered backprojection algorithm: The filtering part, which can be visualized as a simple weighting of each projection in the frequency domain, and the backprojection part, which is equivalent to finding the elemental reconstructions corresponding to each wedge filter mentioned above.

The first step mentioned above accomplishes the following: A simple weighting in the frequency domain is used to take each projection and estimate a pie-shaped wedge of the object's Fourier transform. Perhaps the simplest way to do this is to take the value of the Fourier transform of the projection, $s_\theta(\omega)$, and multiply it by the width of the wedge at that frequency. Thus if there are K projections over 180° then at a given frequency ω, each wedge has a width of $2\pi|\omega|/K$.

The effect of this weighting by $2\pi|\omega|/K$ is shown in Figure 3-1(c). Comparing this to that shown in (a) we see that at each spatial frequency, ω, the weighted projection, $(2\pi|\omega|/K)s_\theta(\omega)$, has the same "mass" as the pie-shaped wedge. Thus the weighted projections represent an approximation to the pie shaped wedge but the error can be made as small as desired by using enough projections.

The final reconstruction is found by adding together the two-dimensional inverse Fourier transform of each weighted projection. Because each projection only gives the values of the Fourier transform along a single line, this inversion can be performed very quickly. This step is commonly called a backprojection since, it can be perceived as the smearing of each filtered projection over the image plane.

The complete filtered backprojection algorithm can therefore be written as:

* Sum for each of the K angles, θ, between 0 and 180°

* Measure the projection, $P_\theta(t)$
* Fourier transform it to find $s_\theta(\omega)$
* Multiply it by the weighting function $2\pi|\omega|/K$
* Sum over the image plane the inverse Fourier transforms of the filtered projections (the backprojection process).

There are two advantages to the filtered backprojection algorithm over a frequency domain interpolation scheme. Most importantly, the reconstruction procedure can be started as soon as the first projection has been measured. This can speed up the reconstruction procedure and reduce the amount of data that must be stored at any one time. To appreciate the second advantage, it should be noted that in the filtered backprojection algorithm, when we compute the contribution of each filtered projection to an image point, interpolation is often necessary; it turns out that it is usually more accurate to carry out interpolation in the space domain, as part of the backprojection or smearing process, than in the frequency domain. Simple linear interpolation is often adequate for the backprojection algorithm while more complicated approaches are needed for direct Fourier domain interpolation.

3.1.2. Theory

The derivation of the backprojection algorithm for a parallel beam geometry begins with the inverse Fourier transform: [7]

$$f(x,y)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}F(u,v)e^{j2\pi(ux+vy)}dudv \qquad (3.1)$$

Converting to polar coordinates and substituting the Fourier transform of the projection at angle θ, $s_\theta(\omega)$, for the two-dimensional Fourier transform $F(\omega,\theta)$, we get

$$f(x,y)=\int_{0}^{\pi}Q_\theta(x\cos\theta+y\sin\theta)d\theta \qquad (3.2)$$

where

$$Q_\theta(t) = \int_{-\infty}^{+\infty} s_\theta(\omega)|\omega| e^{j2\pi\omega t} d\omega \qquad (3.3)$$

This estimate of f(x,y), given the projection data transform $s_\theta(\omega)$, has a simple form. Equation (3.3) represents a filtering operation, where the frequency response of the filter is given by $|\omega|$; therefore $Q_\theta(t)$ is called a "filtered projection." The resulting projections for different angles θ are then added to form the estimate of f(x,y).

Equation (3.2) calls for each filtered projection, Q_θ, to be "backprojected." This can be explained as follows. To every point (x,y) in the image plane there corresponds a value of $t = x\cos\theta + y\sin\theta$ for a given value of θ , and the filtered projection Q_θ contributes to the reconstruction its value at t $(= x\cos\theta + y\sin\theta)$. This is further illustrated in Figure 3-2. It is easily shown that for the indicated angle θ , the value of t is the same for all (x,y) on the line LM. *Therefore, the filtered projection, Q_θ, will make the same contribution to the reconstruction at all of these points.* Therefore, one could say that in the reconstruction process each filtered projection, Q_θ, is smeared back, or backprojected, over the image plane.

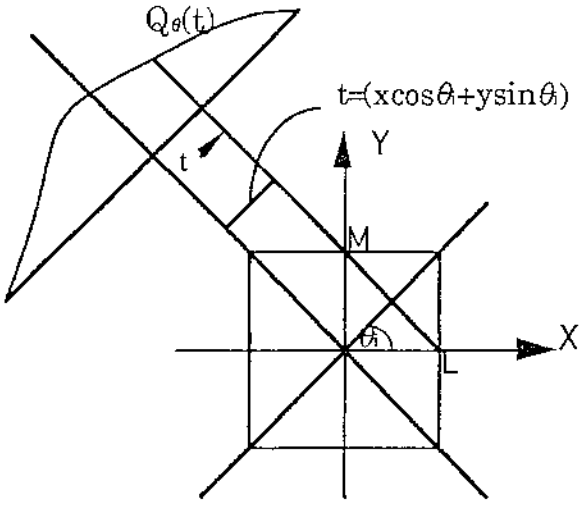


Fig. 3-2 Backprojection reconstruction

The equation (3.2) can be further changed to

$$Q_\theta(t) = \int_{-W}^{+W} s_\theta(\omega)|\omega| e^{j2\pi\omega t} dw \approx \frac{2W}{N} \sum_{m=-N/2}^{N/2} s_\theta(m\frac{2W}{N}) \left| m\frac{2W}{N} \right| e^{j2\pi m(\frac{2W}{N})t} \qquad (3.4)$$

where the frequency domain is assumed bandlimited and the projection data is assumed to be zero for large values of |t|.

However the requirement of finite order and finite bandwidth will lead to artifacts in the reconstructed image. The artifacts can be eliminated by the following alternative implementation of (3.3) which doesn't require the approximation used in the discrete representation of (3.4). When the highest frequency in the projections is finite, (3.3) may be expressed as

$$Q_\theta(t) = \int_{-\infty}^{+\infty} s_\theta(\omega) H(\omega) e^{j2\pi\omega t} d\omega \qquad (3.5)$$

where

$$H(\omega) = |\omega| b_W(\omega) \qquad (3.6)$$

where, again,

$$b_W(\omega) = \begin{cases} 1 & |\omega| < W \\ 0 & \text{otherwise} \end{cases} \qquad (3.7)$$

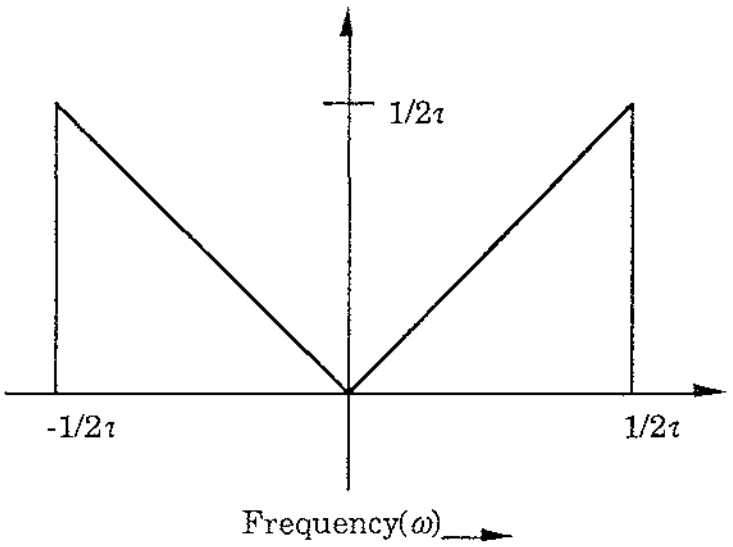


Fig. 3-3 The ideal filter response for the filtered backprojection algorithm is shown here. It has been bandlimited to $1/2\tau$.

$H(\omega)$, shown in Figure 3-3, represents the transfer function of a filter with which the projections must be processed. The samples of impulse response, $h(n\tau)$, of this filter is given by the inverse Fourier transform of $H(\omega)$ and is

$$h(n\tau) = \begin{cases} 1/4\tau^2, & n = 0 \\ 0, & n \text{ even} \\ -\dfrac{1}{n^2\pi^2\tau^2}, & n \text{ odd} \end{cases} \qquad (3.8)$$

This function is shown in Figure 3-4.

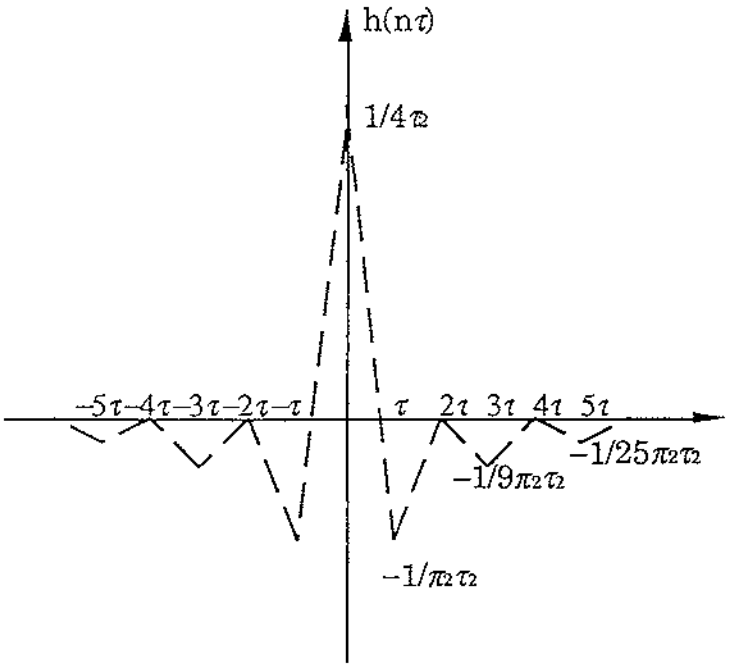


Fig. 3-4 The impulse response of the filter shown in Fig. 3-3

Since both $P_\theta(t)$ and $h(t)$ are now bandlimited functions, they may be expressed as

$$P_\theta(t) = \sum_{k=-\infty}^{\infty} P_\theta(k\tau)\frac{\sin 2\pi W(t-k\tau)}{2\pi W(t-k\tau)} \tag{3.9}$$

$$h(t) = \sum_{k=-\infty}^{\infty} h(k\tau)\frac{\sin 2\pi W(t-k\tau)}{2\pi W(t-k\tau)} \tag{3.10}$$

By the convolution theorem the filtered projection (3.5) can be written as

$$Q_\theta(t) = \int_{-\infty}^{\infty} P_\theta(t')h(t-t')dt' \tag{3.11}$$

Substituting (3.9) and (3.10) in (3.11) we get the following result for the values of the filtered projection at the sampling points:

$$Q_\theta(nt) = \tau \sum_{k=-\infty}^{\infty} h(n\tau - k\tau)P_\theta(k\tau) \tag{3.12}$$

In practice each projection is of only finite extent. Suppose that each $P_\theta(k\tau)$ is zero outside the index range k=0, ... ,N-1. We may now write the following two equivalent forms of (3.12):

$$Q_\theta(n\tau) = \tau \sum_{k=0}^{N-1} h(n\tau - k\tau) P_\theta(k\tau), \quad n = 0,1,2,\ldots,N-1 \tag{3.13}$$

or

$$Q_\theta(n\tau) = \tau \sum_{k=-(N-1)}^{N-1} h(k\tau) P_\theta(n\tau - k\tau), \quad n = 0,1,2,\ldots,N-1 \tag{3.14}$$

The discrete convolution in (3.13) or (3.14) may be implemented directly on a general purpose computer.

3.2. Reconstruction Algorithm for Fan Projections

The theory in the preceding section dealt with reconstructing images from their parallel projections. In generating these parallel data a source-detector combination has to linearly scan over the length of a projection, then rotate through a certain angular interval, then scan linearly over the length of the next projection, and so on. This usually results in times that are as long as a few minutes for collecting all the data. A much faster way to generate the line integrals is by using fan beams such as those shown in Figure 3-5. One now uses a point source of radiation that emanates a fan-shaped beam. On the other side of the object a band of detectors is used to make all the measurements in one fan simultaneously. The source and the entire bank of detectors are rotated to generate the desired number of fan projections. As might be expected, one has to pay a price for this simpler and faster method of data collection; as we will see later the simpler backprojection of parallel beam tomography now becomes a weighted backprojection. There are two types of fan projections depending upon whether a projection is sampled at equiangular or equispaced intervals. Because the reconstruction algorithm for these two types of fan projection is similar, we only discuss the equiangular here.

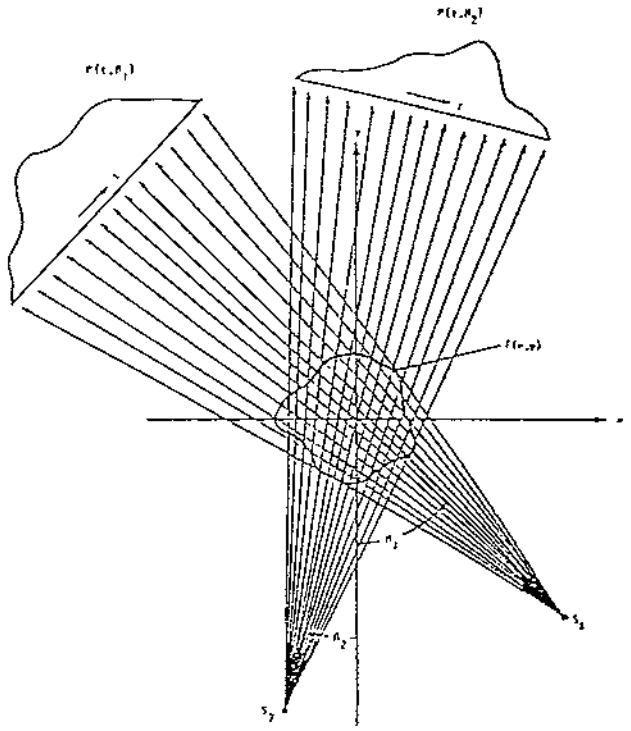


Fig. 3-5. A fan beam projection is collected if all the rays meet in one location.

According to the [7], The final equation for fan beam geometry to reconstruct an image is:

$$f(r,\phi)=\frac{1}{2}\int_0^{2\pi}\frac{1}{L^2}Q_\beta(\gamma')d\beta \tag{3.15}$$

where

$$Q_\beta(\gamma)=R'_\beta(\gamma)*g(\gamma) \tag{3.16}$$

and where

$$R'_\beta(\gamma)=R_\beta(\gamma)\cdot D\cdot\cos\gamma \tag{3.17}$$

Here β is the angle that the source S makes with a reference axis, and angle γ gives the location of a ray within a fan, and D is the distance of the source S from the origin O.

This calls for reconstructing an image using the following three steps:

Step 1:

Assume that each projection $R_\beta(\gamma)$ is sampled with sampling interval α. The known data are then $R_{\beta_i}(n\alpha)$ where n takes integer values and β_i are the angles at which projections are taken. The first step is to generate for each fan projection $R_{\beta_i}(n\alpha)$ the corresponding $R'_{\beta_i}(n\alpha)$ by

$$R'_{\beta_i}(n\alpha) = R_{\beta_i}(n\alpha) \cdot D \cdot \cos n\alpha \tag{3.18}$$

Note that n=0 corresponds to the ray passing through the center of the projection.

Step 2:

Convolve each modified projection $R'_{\beta_i}(n\alpha)$ with $g(n\alpha)$ to generate the corresponding filtered projection:

$$Q_{\beta_i}(n\alpha) = R'_{\beta_i}(n\alpha) * g(n\alpha) \tag{3.19}$$

Step 3:

Perform a weighted backprojection of each filtered projection along the fan. The backprojection here is very different from that for the parallel case. For the parallel case the filtered projection is backprojected along a set of parallel lines as shown in Figure 3-6 (a). For the fan beam case the backprojection is done along the fan (Fig. 3-6 (b)).

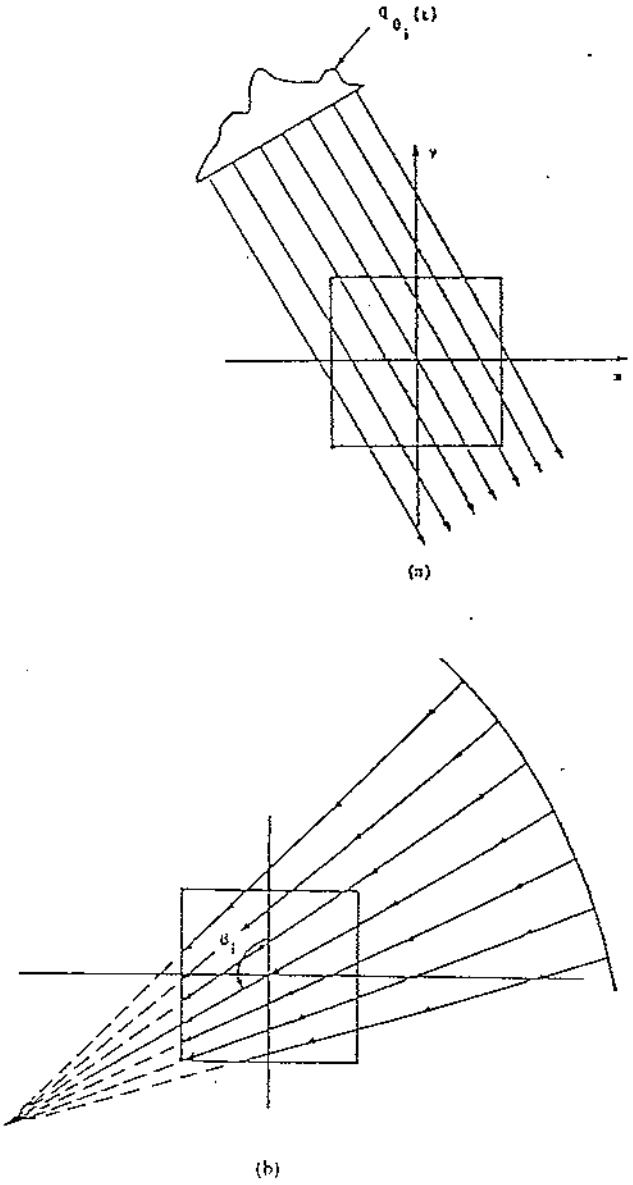


Fig. 3-6. Fan and parallel beam projections.

While the filtered projections are backprojected along parallel lines for the parallel beam case (a), for the fan beam case the backprojection is performed along converging lines (b).

This concludes our presentation of the algorithm for reconstructing projection data measured with detectors spaced at equiangular increments.

From the presentation, it is obvious that the reconstruction algorithm for fan beam projection is similar to the parallel projection, so we will only consider the parallel projection.

3.3. Modified Backprojection Algorithm

From section 3.1, we know that the standard reconstruction equations are

$$f(x,y)=\frac{1}{2}\int_0^{2\pi}\int_{-\infty}^{\infty} Q(\theta,t)\cdot\delta(x\cos\theta+y\sin\theta-t)\,dt\,d\theta \tag{3.20}$$

$$Q(\theta,t)=\int_{-\infty}^{\infty} P(\theta,\alpha)g(t-\alpha)\,d\alpha \tag{3.21}$$

Equation (3.21) convolves the parallel projection P with an anti-blurring kernel g and equation (3.20) performs the backprojection of the filtered projection Q over the reconstruction area f using the mapping function δ. The name *mapping function* is adopted because it maps the projection functions onto the reconstruction area.

The measurement hardware provides a finite number of projections N at equi-angular displacements around the object, each containing M rays, and the reconstruction area is divided up into an array of rectangular pixels (Cartesian pixel geometry). Incorporating these into equation (3.20) gives the discrete backprojection equation

$$f(l\Delta x,k\Delta y)=\frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta,m\Delta t)\delta(l\Delta x\cos(n\Delta\theta)+k\Delta y\sin(n\Delta\theta)-m\Delta t) \tag{3.22}$$

where l=1,2,3, ..., L. k=1,2,3, ..., K.

An interpolating mapping function δ_I is now required because the backprojection of any ray value $Q(n\Delta\theta,m\Delta t)$ will in general not coincide exactly with any of the discrete pixels $f(l\Delta x,k\Delta y)$. The interpolation of the

function δ_i may be nearest-neighbour, linear, or other more elaborate interpolation scheme. However linear and nearest-neighbour are the most common due to their low computational requirement.

An obvious implementation of equation (3.22) would be to pre-calculate the values of the mapping function and store them in a look-up table for subsequent retrieval at run-time. For the standard algorithm using nearest-neighbour interpolation, this requires $N \times W$ elements in the table (where N is the number of projections, and W is the number of pixels in the reconstruction area).

It is possible to reduce the size of the mapping function by a factor of N by discretising the reconstruction area using a non-Cartesian coodinate system. In order to develop an algorithm which will achieve this reduction, the coordinates of the standard backprojection equation (3.20) are transformed from Cartesian (x,y) to polar (r,ϕ).

$$f(r,\phi) = \frac{1}{2}\int_{0}^{2\pi}\int_{-\infty}^{\infty} Q(\theta,t)\delta(r\cos[\phi - \theta] - t)dtd\theta \qquad (3.23)$$

Equation (3.23) describes the system as continuous in both the reconstruction area and the projection space. However the projection space is discrete by nature of the physical hardware in both the ray position t and the projection angle θ. Thus equation (3.23) becomes

$$f(r,\phi) = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_i(r\cos[\phi - n\Delta\theta] - m\Delta t) \qquad (3.24)$$

An interpolating mapping function δ_i is required because the filtered projection data is only available at discrete points, although the reconstruction space f is still described as continuous. The reconstruction area can be divided into N equal angular segments of size $\Delta\theta$ where $\Delta\theta$ is the angular displacement between projections. The angle ϕ in equation (3.24) can be replaced by $j\Delta\theta + \phi'$ where j indicates the segment and ϕ' indicates the angular offset within the segment such that $0 \le \phi' < \Delta\theta$,

$$j = \text{truncation}(\frac{\phi}{\Delta\theta}), \quad \phi' = \text{modulo}(\phi, \Delta\theta) \tag{3.25}$$

Equation (3.24) becomes

$$f(r, j\Delta\theta + \phi') = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_I(r\cos[(j-n)\Delta\theta + \phi'] - m\Delta t) \tag{3.26}$$

Since $(j-n)\Delta\theta$ is azinuthally periodic it may be replaced by the expression module $((j-n),N)\Delta\theta$, and by defining the mapping function for values j=0, ..., N-1 at one angle $n\Delta\theta$, the function δ_I is defined for all projection angles $n\Delta\theta$, n=1, ..., N. The general modified backprojection equation is

$$f(r, j\Delta\theta + \phi') = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_I(r\cos[\text{mod}[(j-n),N]\Delta\theta + \phi'] - m\Delta t)$$

(3.27)

The mapping function δ_I in equation (3.27) is reduced by a factor of N from equation (3.26).

To implement equation (3.27) using a digital computer it is necessary to discretise the reconstruction area f. Since most current tomographic systems require a large number of projections, the number of sectors in the reconstruction area will also be large (because the number of sectors is determined by the number of projections N). It is reasonable then to discretise the rotational component in equation (3.75) from $j\Delta\theta + \phi'$ to $i\Delta\phi$, where $\Delta\phi = \Delta\theta$ (as the segment angle is the same as the projection increment angle), and the radial component from r to $(h - \frac{1}{2})\Delta r$ where h is an integer greater than zero. This pixel geometry is shown in Figure 3-11 and is described by

$$f([h - \frac{1}{2}]\Delta r, i\Delta\phi) = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_I([h - \frac{1}{2}]\Delta r\cos[(i-n)\Delta\theta] - m\Delta t)$$

(3.28)

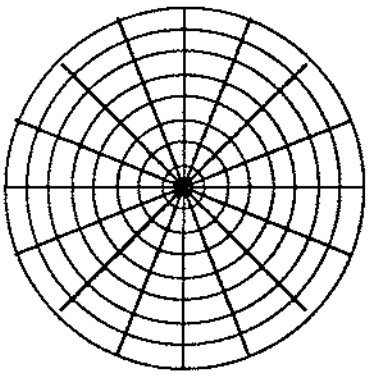


Fig. 3-7. Equi-Radial/Angular Pixel Geometry

To arrange this reconstruction area onto a Cartesian array of elements appropriate for digital processing, we consider the radial component r in the x-direction, and the angular component ϕ in the y direction. Equation (3.28) can now be written

$$f(x,y) = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_1(x\cos(y - n\Delta\theta) - m\Delta t) \qquad (3.29)$$

where x=1,2,3, ..., X; y=1,2,3, ... Y.

Using nearest-neighbour interpolation the mapping function is

$$\delta_1(t) = \delta(\mathrm{rnd}[\frac{t}{\Delta t}]) \qquad (3.30)$$

where rnd indicates the mathematical rounding operation.

Let the mapping function δ_1 be defined for all points (x,y) in the first projection ($n\Delta\theta = 0$). Noting that an x-directional shift in the exponential conformal (x,y) domain corresponds to a rotation in the (r,ϕ) domain about the origin, shifting the mapping function values in the y-direction corresponds to a rotation of the projection over the reconstruction space.

To maximise the rate at which the algorithm can be processed, equation (3.29) must be broken down into small computational blocks which can all process information at the same time. In this form it is easy to implement on a systolic array, where each block makes up one element of the array.

The modified algorithm is well suited to pre-calculated table look-up techniques since the number of values in the mapping function are reduced by a factor of N (the number of projections), leaving the number of values equal to the number of pixels ($X \times Y$).

Defining the look-up table as $\gamma(x,y)$, equation (3.29) may be expressed as:

$$f(x,y) = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_I(\gamma(x,y) - m\Delta t) \qquad (3.31)$$

By performing the round operation on the pre-calculated values, the argument in the mapping function will be integer, and mapping function δ_I becomes the dirac function δ.

To separate the algorithm into a large number of cells suitable for systolic array organisation, each pixel (x,y) becomes an individual process. From equation (3.31) it is clear that each process involves selecting the correct value of Q corresponding to the appropriate γ value. This value is then added to the pixel value. To implement this process each cell must contain:

(1) Storage for the image cell $f(e^x, y)$

(2) Storage for the filtered ray value $Q(n\Delta\theta, m\Delta t)$

(3) Storage for the pre-calculated mapping function argument value $\gamma(x,y)$

(4) A counter m to increment the ray number $m\Delta t$

(5) A comparitor to determine when the dirac argument is zero

(6) An adder to add the appropriate ray values to the image cell

After the first projection, calculation of the look-up table can be performed by circular shifting the γ values in the y-direction of the array after each projection.

Figure 3-8 shows a block diagram of the hardware for a single block of the systolic array.

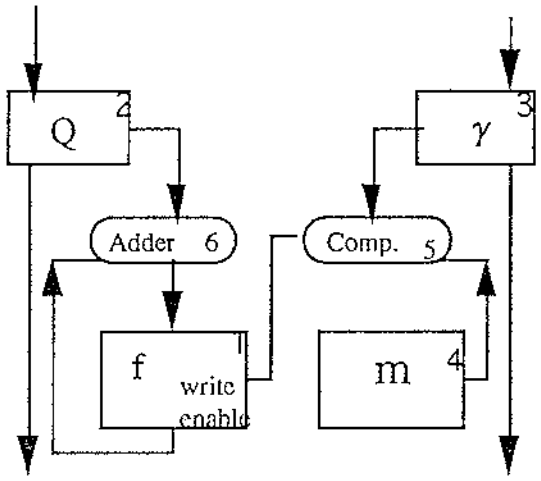


Fig. 3-8 Block diagram of the hardware for a single block of the systolic array

The filtered data Q is passed through the array by the block interconnections at regular synchronised time intervals. Each filtered ray value is passed into the storage cell Q for one cycle only, whereas the look-up values remain in the storage cell γ for 2M cycles (1 projection duration).

Figure 3-9 shows how these blocks are arranged in the systolic array.

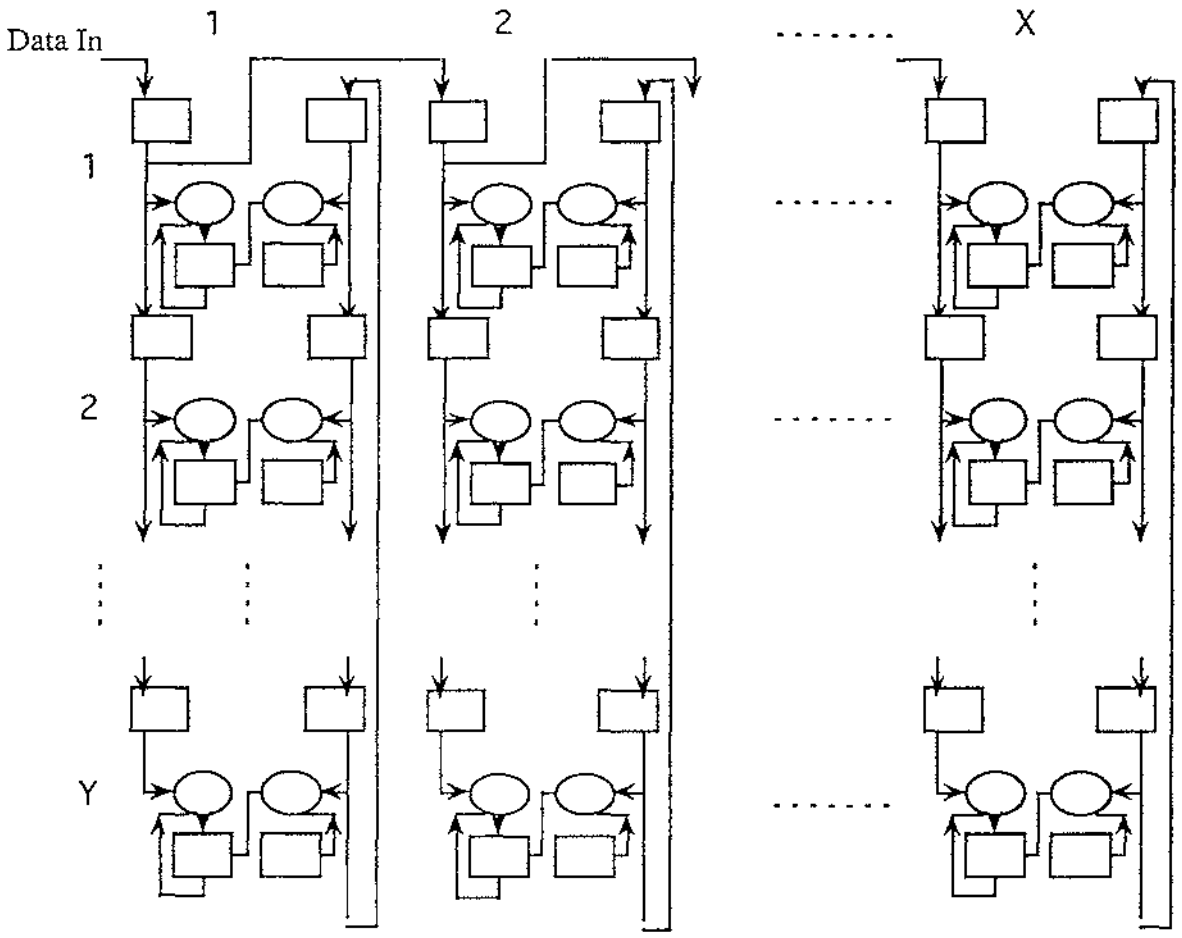


Fig. 3-9 The blocks in systolic arrays

Note from Figure 3-9 that there are three variations on the basic cell.

(1) The top left cell is the only cell to receive filtered data Q from outside the system. The data is passed to the cell below, and the cell to the right.

(2) The top row of cells receive ray data from the cell to the left, and pass the ray value to the right, as well as down to the cell below.

(3) The bottom row of cells do not pass the ray data on.

Because of the cell interconnections the data is rippled through the array from top left to bottom right moving one cell per cycle. Processing is parallel going from bottom left to top right (since all diagonal cell process the same

ray value Q), and pipelined going from top left to bottom right. All projection data moves through the array in unison, so registers can be used to shift the ray data. However since the γ values all move at once and only when the cell has processed an entire projection, a temporary storage unit γ' is required to store the old γ value before the new γ value is written. Without this the old γ value would be overwritten before being shifted to the cell below. The cell layout now becomes

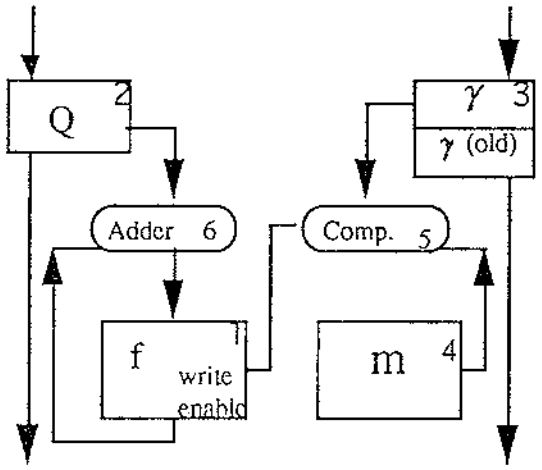


Fig. 3-10 The new block diagram of the hardware for a single block of the systolic array

Note that Y is equal to the number of projections N, so the top cell at position X will be ready to process the first ray of projection n while the bottom cell at position x is processing the first ray of projection n-1. Therefore it is possible for the array of cells to be processing multiple rays from multiple projections at the same time.

For implementation purposes it is noted that the motion of the γ values with respect to the pixel values f is relative, and it is advantageous to keep the γ values stationary whilst moving the pixel values f through the array after each projection. This simplified the retrieval of the reconstructed image because after completion the pixels may be shifted out.

CHAPTER 4. PARALLEL PROCESSOR IMPLEMENTATION IN COMPUTERIZED TOMOGRAPHY USING TRANSPUTERS

To reconstruct an image in real time, one solution is to use Parallel Processing. Normally, the time spent in tomographic reconstruction is very long. According to the algorithms introduced in the last chapter, in which we simulated the reconstruction of a simple image in a general PC machine, the reconstruction time is about 25 minutes. In this chapter, we discuss how transputers are used to reconstruct such an image in a much shorter time.

4.1. Analysis of the Reconstruction Algorithms

The procedure we use to reconstruct an image is:

(1) Sample the projection data $P_\theta(k\tau)$, k=0,1, ..., N-1. Where τ is the projection data spatial sampling interval.

(2) Use the equation

$$Q_\theta(n\tau) = \sum_{k=0}^{N-1} h(n\tau - k\tau) P_\theta(k\tau), \quad n=0,1,2, ..., N-1 \tag{4.1}$$

$$\text{where } h(n\tau) = \begin{cases} 1/4\tau^2, & n = 0 \\ 0, & n = \text{even} \\ -\dfrac{1}{n^2\pi^2\tau^2}, & n = \text{odd} \end{cases}$$

to generate the filtered projection data $Q_\theta(n\tau)$.

(3) In polar coordinates, the reconstructed image f(x,y) can be obtained using the modified backprojection algorithm of equation (3.31).

$$f(x,y) = \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=-M}^{M} Q(n\Delta\theta, m\Delta t)\delta_1(\gamma(x,y) - m\Delta t) \tag{4.2}$$

Note that, x represents the radial component r, y represents the angular component ϕ.

(4) In order to reconstruct an image to display in a Cartesian coordination, we need to interpolate to obtain the X-Y data. The nearest neighbour interpolation method is used here.

(5) The X-Y data of step 4 is used to display the reconstructed image.

During the simulation, the steps 1 and 5 are not part of the image reconstruction, so they were done in the general PC machine. Steps 2, 3 and 4 were done in the transputer network.

According to the analysis of Chapter 2, if we want to use a transputer network, we should first consider the Decomposability, Complexity of the algorithm and its Communication requirement.

Decomposition.

Let us study the steps 2, 3 and 4 in detail. It is obvious that we can execute the image reconstruction in a polar coordination only after filtered projection data are obtained. This means that steps 2 and 3 are sequential and can not be executed in parallel. Step 4, must also be executed after step 3 because only after we have the reconstruction image in polar coordination, can we interpolate them to a Cartesian coordination. In general, steps 2, 3 and 4 must be executed in sequence, they can be computed in parallel only for mutiple reconstructions and with suitable synchronisaton. But how about the individual steps, can they be internally executed in parallel?

For step 2, the projection data coming in from step 1 are view by view, each view consists of certain rays, i.e. projection data $P_\theta(k\tau)$, k=0, 1, ... , N-1 can be rewritten as $P_\theta[j][i]$, i=0, 1, ... , (rays-1); j=0, 1, ... , (views-1), rays $\times$ views=N. From the equation (4.1), convolved data can be obtained.

$$Q_\theta[j][i] = \sum_{r=0}^{rays-1} h(i-r)P_\theta[j][r] \ , \quad i=0, 1, \ldots, (rays-1); j=0, 1, \ldots, (views-1) \tag{4.3}$$

From equation (4.3), we can see that it is possible to decompose using either the rays or views because they do not have a sequential relationship. Since the projection data is coming in ray by ray, it is best to decompose the rays. So step 2 can be used in a transputer network.

For step 3, from the chapter 3, we know that the reconstruction image f(x,y) is given by:

$$f(x,y) = \int_0^\pi Q_\theta(x\cos\theta + y\sin\theta)d\theta$$

Q_θ is a convolved projection at angle θ. The equation's meaning is that the reconstruction image f(x,y) is formed by the integration of the values of the convolved projections associated with all rays passing through the point (x,y). This is the normal algorithm. In the modified algorithm, the reconstruction area is divided into sections and rings, every block represents a reconstruction point (x,y), so the more sections and rings, the higher resolution. When convolved data arrives view by view, every block records a component of the rays passing through them, the sum of these ray components is the reconstruction value of that point. In the reconstruction area, it is obvious that every reconstruction pixel value calculation is the same, so if we have enough transputers in the network, each transputer can execute a pixel reconstruction. If we have not enough transputers in the network because the number of image pixels is very large, we can group them together, then each transputer can deal with a different group of pixels. In general, it is possible to decompose the backprojection step.

For step 4, in order to display the reconstructed picture, the nearest neighbour interpolation method is used to translate the point from polar coordination to Cartesian coordination. Because the Cartesian reconstruction area was defined as a square, it can be decomposed using the X and Y-axes. The X-axis or Y-axis

can be divided into parts, each transputer calculates a different part, then connects them together to get the value of the whole reconstructed image.

From the above analysis, the steps 2, 3 and 4 must be in sequence, but internally step 2 itself can be executed in parallel, as can steps 3 and 4.

Complexity.

How effective is the divide-and-conquer strategy in obtaining the maximum gain in speed when several processors are used to process the decomposed tasks? To answer this question one should study how well a problem can be partitioned for solution by a given computer architecture or by an algorithm. For instance, let T be the total time taken by an algorithm in which there is a serial portion taking time T_s and a parallel portion taking time T_p, so that $T_s + T_p = T$; then, no matter how many processors are used, the serial portion would limit the increase in speed by a factor of at most T / T_s; this is because at best T_p can go down to near zero! Thus, if an algorithm has a 10% serial portion, the speed increase that can be achieved by putting infinitely many processors to work would still be limited to ten times.

Therefore a judicious choice is needed in partitioning an algorithm and minimizing the serial portion.

Studying step 2, from equation (4.3), we see that every convolved data Q[j][i] has the same calculation equation, no matter which transputer it is used in. Most of the transputers in the network would have the same programming and arrangement. This satisfies our programming requirements.

Similarly for step 3, the modified reconstruction algorithm is the same for every reconstructed pixel, when they (pixels) are grouped together to be executed in a transputer, the transputer processing procedure is the same. So the backprojection can not only be decomposed but also suitable for use in a transputer network.

For step 4, interpolation to Cartesian system, when the X-axis is decomposed, every group in a transputer has the same interpolation calculation equation, so it is suitable to use in transputer network.

Communication.

When a large problem is broken into smaller tasks, it is necessary to organise the coordination between these tasks. To achieve this coordination we need communication links between the different tasks. The larger the number of pieces a problem is split into, the more communication links the resulting algorithm requires. In fact, the number of directed two-way communication links between any two tasks among n tasks could be n(n-1) and so the communication complexity grows quadratically. The communication problem introduces a new dimension to parallel programming and parallel architecture design. This means that in addition to the computing time and memory space requirements of an algorithm, we must also consider the communication costs and set a bound on the complexity of communication. This would imply that communication between tasks which are not close enough (closeness being measured by a suitably defined criterion for an architecture) should be avoided; that is, the communication should preferably be confined to only those processes that are very close neighbours.

When a large number of processors are assigned to carry out the split tasks, it is possible that simultaneous request or access to certain data or tasks may create a conflict or collision. This could slow down the anticipated speed advantage resulting from the multiple processors, or may even lead to a state of inactivity or standstill when two processes or processors wait for each other indefinitely (deadlock), or may delay some process indefinitely (lockout). Since the computational speeds for different tasks are unpredictable (non-deterministic), the different process may become unsynchronized, leading to a total breakdown of the tasks.

The communication problem is therefore concerned with the minimisation of communication complexity, prevention of deadlocks and improved coordination.

Communication is important in parallel processing. We will discuss the communication problem of our reconstruction algorithm in section 4.4 after the transputer we used and the structure of our transputer network are introduced.

4.2. IMS T414 Transputer

The IMS T414 transputer is one of the transputer family products of the INMOS company. It is a 32 bit CMOS microcomputer with 2 Kbytes of on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages on a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond. The IMS T414 provides high performance arithmetic and microcode support for floating point operations. A device running at 20 MHZ achieves an instruction throughput of 10 MIPS.

The basic blocks of the IMS T414 are shown in Figure 4-1.

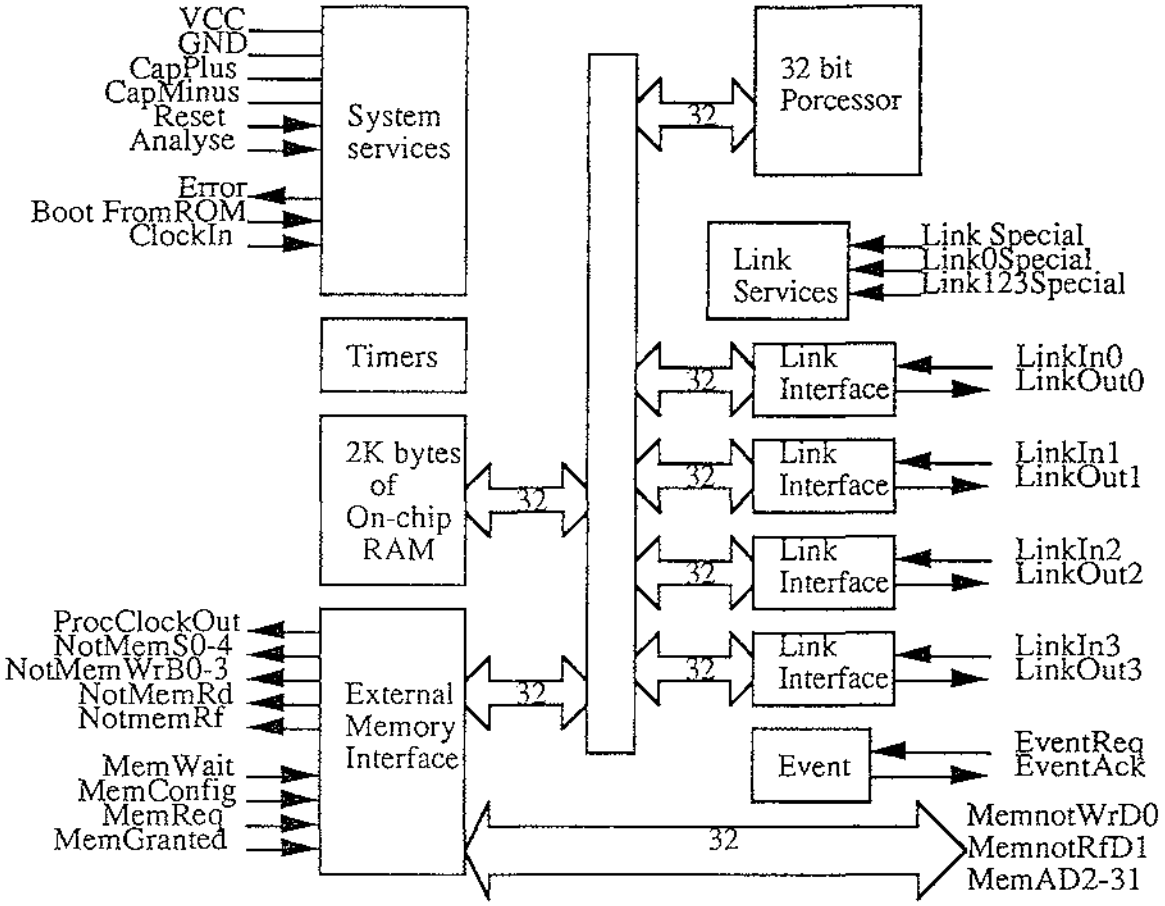


Fig. 4-1. IMS T414 block diagram

The IMS T414 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 150 nanoseconds (26.6 Mbytes/sec) for a 20 MHz device. A configurable memory controller provides all timing, control and RAM refresh signals for a wide variety of mixed memory systems.

System services include processor reset and bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T414 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec.

4.2.1. Pin Designations

Table 4.1. IMS T414 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
Reset	in	System reset
Error	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Bootstrap from external ROM or from link
DisableInRAM	in	Disable internal RAM
HoldToGND		Must be connected to GND
DoNotWire		Must not be wired

Table 4.2. IMS T414 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWrB0 -3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Table 4.3. IMS T414 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge

Table 4.4. IMS T414 event

Pin	In/Out	Function
LinkIn0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special speed for link 0
Link123Special	in	Select special speed for links 1,2,3

4.2.2. Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 2 Kbyte on-chip memory, which can store data or program where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI). The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed
- On a list waiting to be executed.

Inactive - Ready to input.
- Ready to output.
- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes. Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List Figure 4-2, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

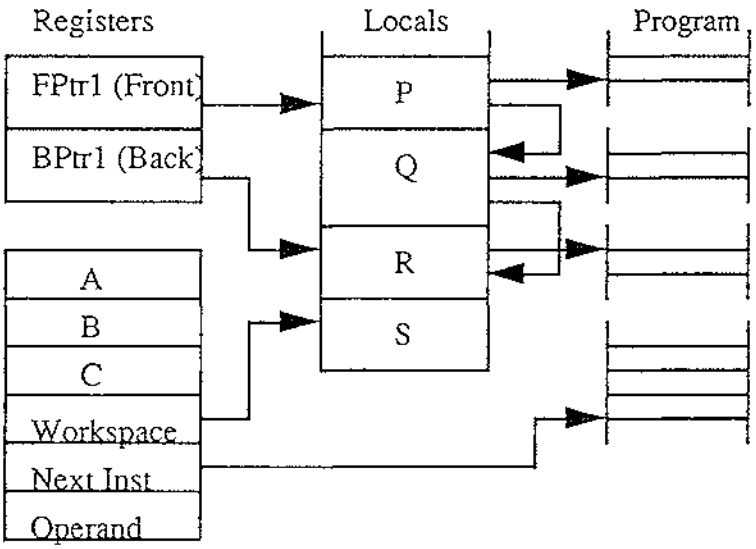


Fig. 4-2. Linked process list

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point. The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1ms apart.

A process can only be descheduled on certain instructions, known as descheduling points. As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1 μs , as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including start process and end process. When a main process executes a parallel construct, start process instructions are used to create the necessary additional concurrent processes. A start process instruction creates a new process by adding a new workspace to the end of the schdeduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the end process instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are started. Each component ends with an end process instruction which decrements and test the counter. For all but the last component, the counter is zero and the main process continues.

4.2.3. Communications

Communications between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs to process queue, no message queue and no message buffer.

A channel between two process executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being *input* message and *output* message.

The *input* message and *output* message instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first become ready must wait until the second one is also ready. A process performs an *input* or *output* by loading the evaluation stack with a pointer to a message, the address of a channel and a count of the number of bytes to be transferred, and then executing an input message or output message instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

4.2.4. Timers

The transputer IMS T414 has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

4.3. The Structure of the Transputer Network

A feature of the INMOS transputer is the bi-directional interprocessor links that allow a transputer to directly communicate with up to four other transputers, thus permitting easy networking. In order to suit image processing, the criteria for selecting an optimum interconnection strategy may be summarised as follows:

(1) The network diameter should be minimised.
(2) The mean interprocessor distance should be minimised.
(3) The system should be incrementally extensible.

(4) Any single faults should not lead to a complete failure of the system.
(5) Algorithms should be able to be implemented as simply as possible.

According to the analysis of R. F. Browne and R .M. Hodgson [25], symmetric chordal networks of degree four offer a practical means of interconnecting processors having four interprocessor links. The symmetry of the networks makes the mapping of algorithms onto the network a comparatively simple task. In particular, for image processing applications, an image will map very simply and conveniently onto the network. In the above paper techniques have been given for determining the diameter and mean interprocessor distance for a given network. It has been shown that if a network is chosen to have a minimum diameter (by appropriate choice of the number nodes and/ or the chord length) then the mean internode distance will also be a minimum. Although the networks are not incrementally extensible, any number of processors can be added to a general chordal ring network. If the number of additional processors is large the optimum chordal length degradation in performance and a minimum of three faults can be tolerated in the network before any nodes become isolated. Although a simple algorithm exists for routing in an optimal chordal ring network, routing in non optimal and broken networks might best be achieved by means of look-up tables.

For a transputer-based image processing system to be effective, there are two particular requirements:

(a) A link must be provided so that a master or host can communicate with the network.

(b) The segmented image should map readily onto the network.

Requirement (a) means that the chordal ring must be broken. Since a pure chordal ring is symmetric the break can be made at any point, and for convenience will be made in the edge connecting nodes N-1 and 0. Requirement (b) depends on the manner in which images are segmented. Since this is commonly carried out in a cellular fashion, chordal rings that are isomorphic to cellular arrays (with appropriate edge connections) are of particular interest. The cellular array is of particular interest in image

processing since rectangular images can be readily segmented in a cellular fashion. Thus the mapping of a segmented image onto a cellular array is conceptually simple, although by no means necessary since a suitable operating system will make all interprocessor communications transparent to the user.

The Production Technology Department's twenty transputer network can be segmented as five rows of four columns, using a chordal displacement of four. Allowing for the break between nodes 43 and 00, the diameter is four, which gives amean distance of 2.120. The corresponding mean distance for the optimum displacement is 2.080. The requirement for cellular mapping thus involves a net efficiency of 98.1%.

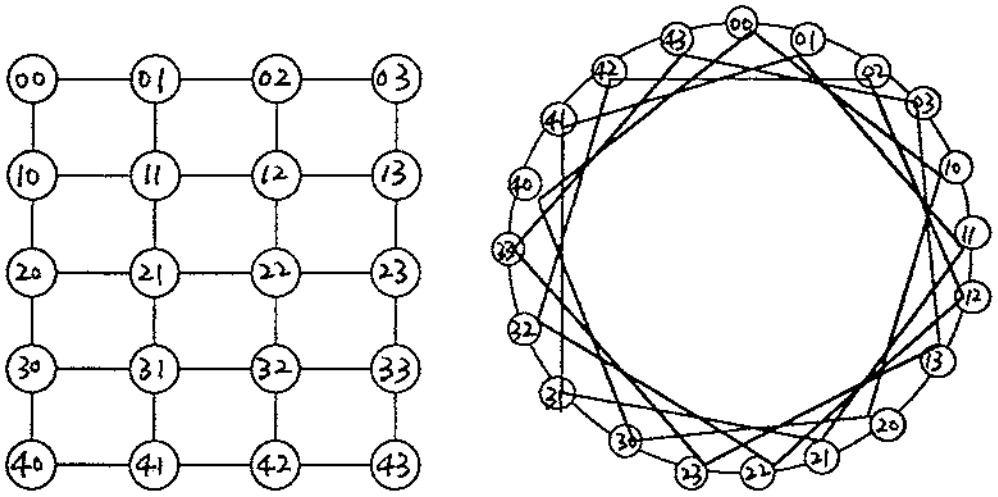


Fig. 4-3 The arrangement of transputer network

4.4. Connectivity of the Transputer System

Every transputer has four communication links. Each link is connected to the main processor via a link interface allowing direct memory access. A transputer can communicate simultaneously on all four links, whilst executing an internal process. Each link implements two occam channels, one for input, the other for output. Figure 4-4 shows the link names and their addresses.

Link1out, for example, is a name chosen by INMOS convention to represent the output channel on link 1.

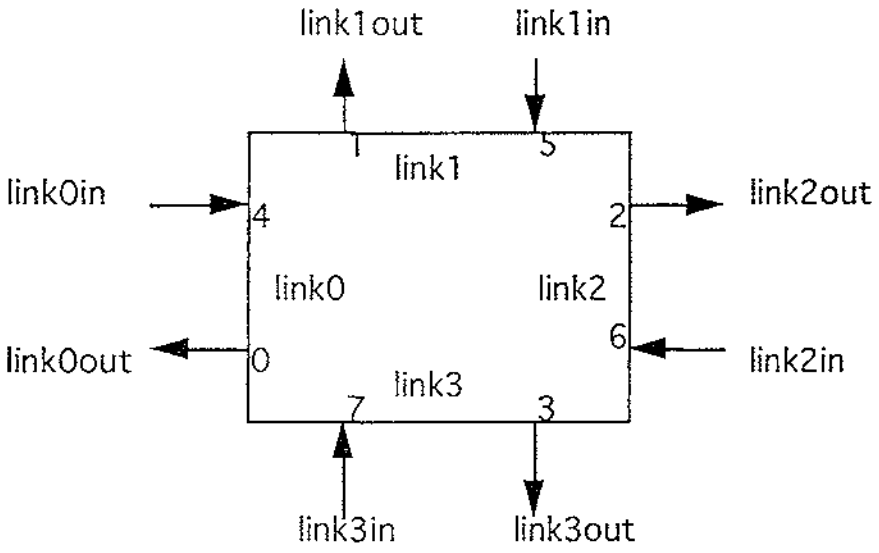


Fig. 4-4 Link names and addresses

Internal Channels (soft channels)

These channels are within a single transputer and are not mapped onto links. Instead, when a channel is defined, a single word of storage is allocated. The two processes which are connected by the channels can access this word. The first one to do so writes a "ready" signal into this word and is then suspended until the second process is ready to communicate. Transfer of data then occurs directly (i.e. not through any real channel) between the processes; the second process continues (it is never suspended), and the first joins the queue of other processes waiting to execute.

External Channels (hard channels)

These are mapped onto the physical links. The link interfaces are responsible for descheduling the process, transferring the entire message (its length in bytes and the data itself) and then rescheduling the process. The transferred message is stored directly in the memory of the receiving transputer without interrupting its processor, thus allowing fast processing.

Two transputers can be linked by connecting them with two one-directional signal wires, along which data is transmitted serially. Each byte of the message must be acknowledged before the next is sent. A byte is transmitted as eleven bits: two leading ones, the eight bits of data and then a zero as a stop bit. The receiving transputer requires only a single-byte buffer, in the link interface, to ensure no information is lost.

4.4.1. Topology of Transputer System

The twenty transputers of Production Technolgy Department are connected as shown in table 4.5.

Table 4.5

Id (No. of transputer)	Link 0	Link 1	Link 2	Link 3
0	host - 2	1 - 0	4 - 3	16 - 2
1	0 - 1	2 - 0	5 - 3	17 - 2
2	1 - 1	3 - 0	6 - 3	18 - 2
3	2 - 1	4 - 0	7 - 3	19 - 2
4	3 - 1	5 - 0	8 - 3	0 - 2
5	4 - 1	6 - 0	9 - 3	1 - 2
6	5 - 1	7 - 0	10 - 3	2 - 2
7	6 - 1	8 - 0	11 - 3	3 - 2
8	7 - 1	9 - 0	12 - 3	4 - 2
9	8 - 1	10 - 0	13 - 3	5 - 2
10	9 - 1	11 - 0	14 - 3	6 - 2
11	10 - 1	12 - 0	15 - 3	7 - 2
12	11 - 1	13 - 0	16 - 3	8 - 2
13	12 - 1	14 - 0	17 - 3	9 - 2
14	13 - 1	15 - 0	18 - 3	10 - 2
15	14 - 1	16 - 0	19 - 3	11 - 2
16	15 - 1	17 - 0	0 - 3	12 - 2
17	16 - 1	18 - 0	1 - 3	13 - 2
18	17 - 1	19 - 0	2 - 3	14 - 2
19	18 - 1	000	3 - 3	15 - 2

Note: 000 indicates that a link is unattached. a-b indicates that a link is attached to transputer a's link b.

In the transputer development system, only the host transputer can contact outside the network. The other transputers in the network can only communicate within the network. According to table 4-5, we see that only transputer 0 can connected to the host computer, so the simplest transputer network consists of the host and transputer 0, as shown in Figure 4-5. This system will be termed the target system.

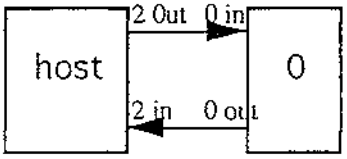


Fig.4-5. Block diagram of target system

The target system was used to test the algorithms and to show the processing time required for a single transputer system.

The inherently sequential nature of the three components of the reconstruction algorithm suggest that the resultant data at each stage should reside in transputer 0 (with connection to the host).

Table 4.5 was used to determine the network size and transputer order for a network structure as shown in figure 4-6. The selection criteria is to select the maximum number of transputers such that the communication connection overhead is minimised, i.e. very close neighbour connection. This led to a 17 transputer network.

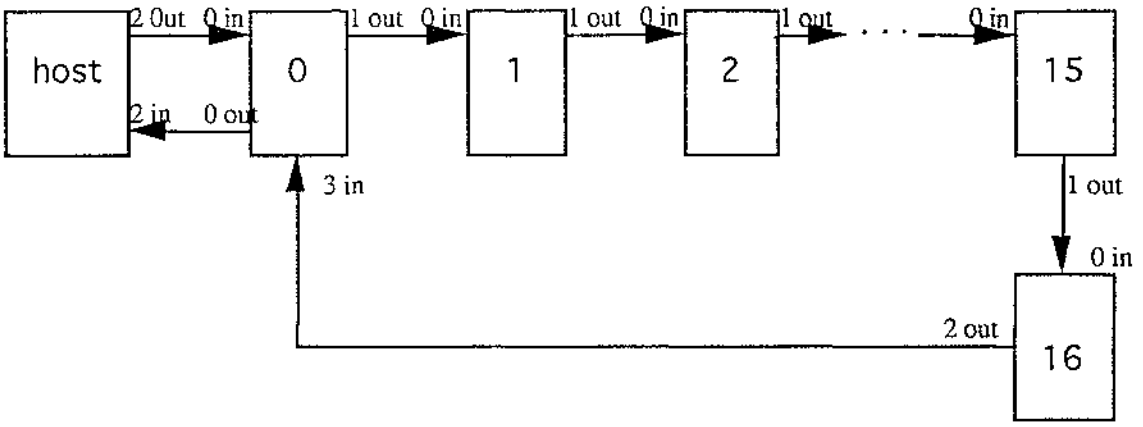


Fig 4-6. The Communication Structure of the Transputer Network

4.5. Algorithm Structure

The full reconstruction algorithm is divided into three components, covolution, backprojection and interpolation. Before we show how they are implemented on the transputer system, we will discuss them again.

4.5.1. Convolution

It is known that the projection data is ray by ray over views, i.e. projection data $P_\theta(k\tau)$, k=0, 1, ... , N-1 can be rewritten as $P_\theta[j][i]$, i=0, 1, ... , (rays-1); j=0, 1, ... , (views-1), rays × views=N. From the equation (4.1), convolved data can be calculated using

$$Q_\theta[j][i] = \sum_{r=0}^{rays-1} h(i-r)P_\theta[j][r] \ , \quad i=0, 1, \ldots , (rays-1); j=0, 1, \ldots , (views-1) \tag{4.4}$$

This FIR filter implements a derivative function and is symmetric with a theoritically optimal length of twice the number of rays in a view. However as the extra data points are achieved by zero padding the restriction of the filter length to just the number of rays in a view reduces the computational effort without significantly reducing the performance of the filter.

4.5.2. Backprojection

According to the modified backprojection algorithm explained in Chapter 3, the backprojection process consists of:

(i) Initialization
This step defines the initialized values of the counter (m), the old γ (γ') and new γ values in every reconstruction block. The reconstruction values F and convolved data Q are zeroed in the beginning state so that they can be dealt with easily in later operations.

(ii) ShiftQ
This step completes the convolved data move according to the algorithm illustrated in Figure 3-9. New convolved data comes in from the top left cell Q[0][0]. The filtered data (convolved data) are passed to the cell below and right. The top row of cells receive ray data from the cell to the left, and pass the ray values to the right, as well as down to the cell below. The bottom row of cells do not pass the ray data on.

(iii) Check γ
If γ=m, then reconstruction value of that point equals F[j][i]+Q[j][i].

(iv) IncM
After finishing the above procedures, counter M should be increased by 1. When one view (one projection) is fully completed, the m values change to zero and a temporary storage unit γ' is required to store the old γ value before the new γ value is written.

4.5.3. Interpolation

After obtaining the backprojection value F[j][i], we have the reconstruction values in polar coordinates. In order to display the reconstructed image, nearest neighbouring interpolation was used to translate from polar coordinates to Cartesian coordinates.

4.6. Implementation Details

The algorithm was first implemented in the target system in order to easily check the correctness of the implementation and to provide timing information when only one transputer is used to solve the reconstruction.

4.6.1. Algorithm Implementation in Target System

The three components of the algorithm were implemented individually, so enabling an assessment of the time taken for each component to be achieved.

i) Convolution

An example is used to explain the implementation. In this example, the projection data consists of 60 projections with 65 rays. Transputer 0 executes the convolution. The host transputer reads the projection data, sending them to the transputer 0 and receiving the convolved data back, then writing them to the controlling PC.

When equation (4.1) is used, the time taken to perform the convolution is 7 minutes 43 seconds, but by using equation (4.4), the convolution time is reduced to only 16 seconds.

ii) Backprojection

Because convolution, backprojection and interpolation are in sequence, after convolution is finished, the host transputer sends the convolved data to transputer 0 to execute the backprojection. The time taken to perform backprojection(Initialization, ShiftQ, Checkγ, and IncM) is 11 minutes.

iii) Interpolation

When the backprojection is completed, if we need to display the image then interpolation will need to be implemented in the transputer target system. The interpolation time is 4 minutes and 48 seconds.

From the above results, we can see that the convolution time is short, but backprojection and interpolation time are much longer, so we should consider the multiple transputer network to reduce the reconstruction time.

4.6.2. Algorithm Implementation in Multiple Transputer Network System

The algorithm was reconfigured for solution on the multiple processors of the transputer network. Once again the three components were developed seperately.

i) Convolution

Because the projection data is projection by projection, it is best to decompose by rays. In our example, there are 65 rays, and for convenience, we arrange transputer 0 to convolve projection data P[j][0], i.e. the first ray was convolved in transputer 0; transputers 1 to 16 execute P[j][i-4 to i], i.e. each transputer executes 4 rays. The communication is:

(a) Transputer 0 convolves P[j][[0], then transmits the convolved data conl[j][0] to transputer 1.

(b) Transputer 1 executes P[j][i], to obtain conl[j][i], (i from 1 to 4), then receive the conl[j][0] from transputer 0, connect them together, and send the connected conl[j][i] (i=0, 1, ..., 4) to transputer 2.

(c) Transputers 2 to 15 are configured similarly to transputer 1.

(d) The end transputer (16) processes P[j][i], i=61, ..., 64, connected data conl[j][i] (i=0,1, ..., rays-1), sending them to transputer 0, ready for backprojection.

The convolution time is 8 seconds. It is obvious that the convolution time is not much shorter, the reason is that communication between the transputers wastes much of the expected time gain.

ii) Backprojection

We know that the communication rule is that the less the communication, the shorter the reconstruction time. From the Figure 3-9, we see the backprojection structure is:

(1) The top left cell is the only cell to receive filtered data Q from the convolution step. The convolved data is passed to the cell below and the cell to the right.

(2) The top row of cells receives ray data from the left cell, and passes the ray value to the right, as well as down to the cell below.

(3) The bottom row of cells do not pass the ray data on.

When the backprojection is run on multiple transputers, it is obvious that we should decompose the backprojection column by column. Since the number of columns is 65, the same as for the convolution step, transputer 0 processes the first column (i=0), the other transputers deal with 4 columns each. According to the structure of Figure 4-5, after obtaining the convolved data conl[j][i], the backprojection processing step is:

(a) In transputer 0, we firstly initialize m, F, Q and γ, then input convolved data to Q[0][0] and output Q[0][0] to transputer 1. With shiftQ, checkγ and incM, finishing this part of the backprojection, sending the reconstruction value F[j][i] (j=0 .. 119, i=0) to transputer 1.

(b) In transputer 1, firstly m, F, Q, γ were initialized. Then data Q[0][0] was received from transputer 0 and Q[0][4] was sent to next transputer. Then shiftQ, checkγ and incM, so finishing this part of the backprojection. Lastly, connecting F[j][i] from transputer 0 and sending the F[j][i] (j=0 .. 119, i=0 .. 4) to the next transputer.

(c) The same procedure is applied to every transputer, then transputer 16 initializes the last dividing part, connecting the whole reconstructed image F[j][i] (i.e. j=0 .. 119, i=0 .. 64), and sending this to transputer 0, so

finishing the complete backprojection and preparing the backprojection data for interpolation.

The time taken to run the backprojection in a 17 transputer network is 1 minutes 3 seconds. This result is very good, the backprojection time is much shorter.

iii) Interpolation

After obtaining the F[j][i], we have the reconstruction value of every block in Figure 4-7. This is in polar coordinates. In order to display the reconstructed image, nearest neighbouring interpolation was used to transform the image from polar coodinates to Cartesian coodinates.

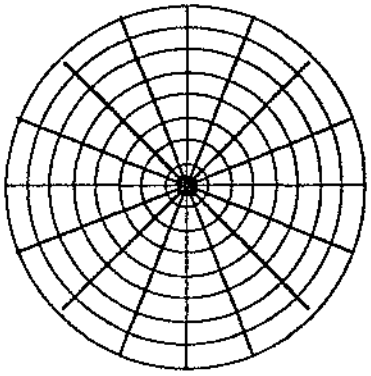


Fig. 4-7 Polar coodinate reconstruction geometry

The Cartesian reconstruction area was defined as the square shown in the Figure 4-8. The reconstruction points in the shadow area were assumed to have a value of 0. By the nearest neighbour interpolation method, every reconstruction point can be obtained. In order to use the transputer network, we should decompose the calculation step. Because we divided the reconstruction area evenly, decomposing the X-axis and Y-axis has the same effect. In my program, the X-axis was divided into several parts, every transputer calculating a different part, then connecting them together to get the whole reconstructed image.

In the example, we divided the whole interpolation into 17 transputers, communication is similar to the convolution and backprojection components. The time taken for interpolation in the transputer network is 48 seconds.

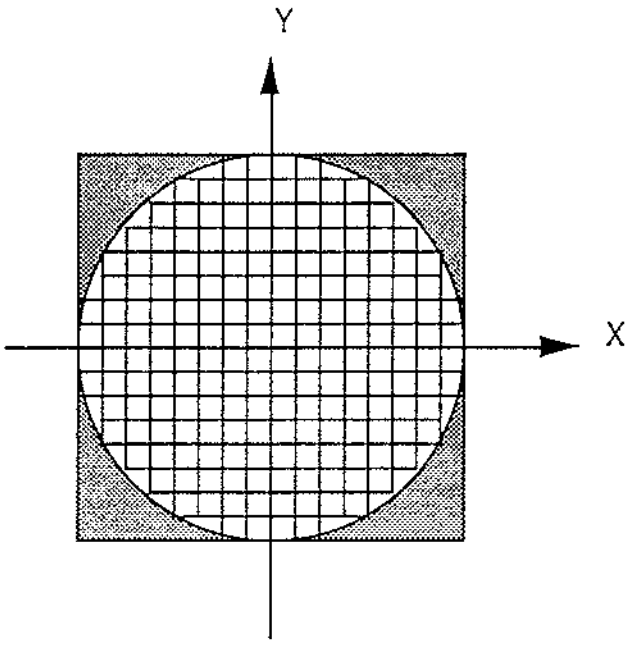


Fig. 4-8 Cartesian reconstruction geometry

iv) Convolution and Backprojection in Transputer Network System

Because interpolation is only necessary to display the image, sometimes the reconstruction value F[j][i] will satisfy our requirements. So we now consider only the convolution and backprojecton together in the transputer network.

We know that the convolution time in the target system is very short, and running it in multiple transputers does not appear to give much advantage, so we arrange transputer 0 to execute the convolution and transputers 1 to 16 to execute the backprojection. The procedure is:

(a) In transputer 0, the projection data P[j][i] were received from host transputer. After they were convolved, the convolution data conl[j][i] were sent to transputer 1.

(b) In transputer 1, convolution data conl[j][i] was stored as Q[0][0] one by one. Transputer 1 executes i=0 to 4, Q[0][4] was sent to transputer 2. Reconstructed values F[j][i] (j=0 to 119, i=0 to 4) were calculated and sent to transputer 2.

(c) In transputer 2, Q[0][4] is firstly received to do backprojection, then F[j][i] (i=5 FOR 4) were obtained, connecting them with F[j][i] (i=0 FOR 5) from transputer 1, we finish with the F[j][i] (j=0 to 119, i=0 to 8), sending them to transputer 3.

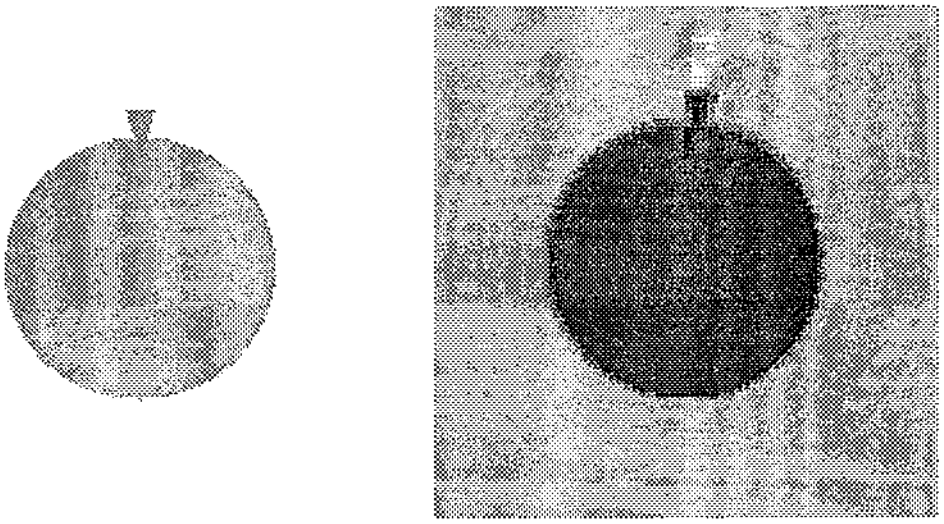
(d) The same as (c) for the other transputers. Transputer 16 receives Q[0][60], for the backprojection, F[j][i] (j=0 to 119, i=61 to 64) were obtained. Connect the F[j][i] from the previous transputer, F[j][i] (j=0 to 119, i=0 to 64) were sent to transputer 0.

(e) Transputer 0 sent F[j][i] (j=0 to 119, i=0 to 64) to the host transputer, then the host transputer tranfered these to the controlling PC. This is our reconstructed value in polar coordinates.

Executing (a) to (e), the time is 1 minute 14 seconds. The transputer network improved the reconstruction time considerably.

v) Convolution, Backprojection and Interpolation in Transputer Network

Finally all these components were connected. Convolution and backprojection were structured as in (iv) above. The polar to cartesian transformation took place sequentially using the entire network. The time taken to perform the reconstruction for 128 by 128 pixels is 1 minute 56 seconds and the reconstructed image and original image are shown in Figure 4-9. It is obvious that the effect of the reconstruction is very good. In order to compare, we simulated the complete reconstruction in the host transputer, the time taken was 22 minutes and 25 seconds. The 1 minute 56 seconds required by the transputer network is a significant reduction in the reconstruction time.



(a) Original image (b) Reconstructed image

Fig. 4-9 Comparison of the reconstructed image with the original image

CHAPTER 5. CONCLUSION

In this thesis, using a parallel processing method, the reconstruction time in the computerized tomography is greatly reduced.

In order to achieve the above reduction, firstly, a modified backprojection algorithm was developed. It makes the standard backprojection equation used in tomographic reconstruction suitable for high-speed table look-up methods.

Secondly, in order to apply tomographic reconstruction in the transputer network, the algorithm structure is analysed. Convolution, backprojection and interpolation are the three components of the algorithm. Although these three components can not be executed in parallel, they can be internally executed in parallel.

A simple structure - the target system was advanced to test the algorithm and show the processing time required for a single transputer system. In order to reduce the reconstruction time as much as possible, the maximum number of transputers in the network should be used and the communication connection overhead should be minimised. According to this criteria, 17 transputers were selected to establish the transputer network.

Finally, an example of 60 projections with 65 rays was used to test the reconstruction result. Firstly, the implementation was used in the target sysem, then the implementation was used in the multiple transputer network system. Five kinds of struture (convolution, backprojection, interpolation, convolution and backprojection, convolution and backprojection and interpolation) were considered. The reconstruction results are good and the reconstruction time is significantly reduced.

REFERENCES:

1. Webb, s.: ' The Physics of Medical Imaging.' Philadelphia, U.S.A. : IOP Publishing Ltd, 1988.

2. Castleman, K. R.: ' Digital Image Processing.' Prentice-Hall, Englewood Cliffs, New Jersey.

3. Jain, A. K.: ' Fundamentals of Digital Image Processing.' U.S.A.: Prentice-Hall, 1989.

4. Scudder, H. J.: ' Introduction to Computer Aided Tomography.' Proceedings of the IEEE, Vol. 66, No. 6, June 1978.

5. Mersereau, R.M. and Oppenheim, A. V.: ' Digital Reconstruction of Multidimensional Signals from Their Projections.' Proceedings of IEEE, Vol. 62, No. 10, October 1974.

6. Herman, G. T.: ' Image Reconstruction from Projections.' New York, U.S.A., Academic Press, 1980.

7. Kak, A. C. and Slaney, M.: ' Principles of Computerized Tomographic Imaging.' New York, U.S.A., IEEE Press, 1988.

8. Offen, R. J.: ' VLSI Image Processing.' London, UK, Collins Professional and Technical Books, 1985.

9. Webber, H. C.: ' Image Processing and Transputers.' Amsterdam, Netherlands, IOS Press, 1992.

10. Carling, A.: 'Parallel Processing: The Transputer and Occam.' England, UK, Sigma Press, 1988.

11. Krishnamurthy E.V.: ' Parallel Processing: Principles and Practice.' Singapore, Addison-Wesley Publishing Company, 1989.

12. Ellison, D.: ' Understanding Occam and the Transputer.' England, U. K., Sigma Press, 1991.

13. Inmos: ' Transputer Databook.' Trowbridge, UK, Redwood Burn Ltd., 1989.

14. Burns, A. ' Programming in Occam 2.' England, UK, Addison-Wesley Publishing Company, 1988.

15. Fleming, P. I. ' Parallel Processing in Control: the transputer and other architectures.' London, UK, Peter Peregrinus Ltd., 1988.

16. Jane, M.R., Fawcett, R. J., Mawby, T.P.: ' Transputer Applications - progress & prospects.' Amsterdam, Nethlands, IOS Press, 1992.

17. Bakkers, A.: ' Applying Transputer Based Parallel Machines.' Amsterdam, Netherlands, IOS Press, 1989.

18. Pritchard, D. J. & Scott, C. J.: ' Applicatiokns of Transputer 2, Proceedings of the Second International Conference on Applications of Transputers. 11-13 July 1990-Southampton, UK.' IOS Press, 1990.

19. Kunii, T. L. and May, D.: ' Transputer / Occam JAPAn3.' Proceedings of the 3rd Transputer / Occam International Conference, 17-18 May 1990 - Tokyo, Japan, IOS Press, 1990.

20. Stiles, G. S.: ' Transputer Research And Applications 1.' Proceedings of the First Conference of the North American Transputer Users Group, April 5-6, 1989 - Salt Lake City, Utah, IOS Press, 1990.

21. Board, J. A.: ' Transputer Research and Applications 2.' Proceedings of the Second Conference of the North American Transputer Users Group, October 18-19, 1989 - Durham, NC, IOS Press, 1990.

22. Wagner, A. S.: ' Transputer Research and Applications 3.' Proceedings of the Third Conference of the North American Transputer Users Group, April 26-27, 1990 - Sunnyvale, CA, IOS Press, 1990.

23. Fielding, D. L.: ' Transputer Research and Applications 4.' Proceeding of the Fourth Conference of the North American Transputer Users Group, October 11-12, 1990 - Ithaca, NY. IOS Press, 1990.

24. Welch, P, Stiles, D. Kunii, T. L. and Bakkers A.: ' Transputing '91.' Proceedings of the World Transputer Use Group Conference, 22-26 April 1991 - Sunnyvale, CA. IOS Press, 1991.

25. Browne, R. F and Hodgson, R. M.: ' Symmetric Degree-four Chordal Ring Networks. ' IEE Proceedings, Vol. 137, Pt. E, No. 4, July 1990.

APPENDIX:

1. This is the exe program for convolution, backprojection and interpolation.

```
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC read.write()
 {{{ hard channel placement values
 VAL link0out IS 0:
 VAL link1out IS 1:
 VAL link2out IS 2:
 VAL link3out IS 3:
 VAL link0in IS 4:
 VAL link1in IS 5:
 VAL link2in IS 6:
 VAL link3in IS 7:
 }}}
 {{{ channel
 CHAN OF ANY app.in, app.out:
 }}}
 {{{ placements
 PLACE app.in AT link2out:
 PLACE app.out AT link2in:
 }}}
 {{{ vals
 VAL pi IS 3.1415 (REAL32):
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 VAL N IS 64:
 VAL b1 IS 0.5(REAL32):
 VAL b2 IS 2.0(REAL32):
 VAL b3 IS 1.0(REAL32):
 VAL b4 IS 0.0(REAL32):
 VAL b5 IS 255.0(REAL32):
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}
 {{{ main program
 PAR
  {{{ read
  [4000]INT ax:
   INT k,j:
  SEQ
```

```
{{{ vals
VAL M IS 3900:
}}}
 {{{ read original data in
 INT input.error:
SEQ
  CHAN OF INT filekeys:
   CHAN OF INT keyboard IS filekeys: -- channel from simulated keyboard
  CHAN OF ANY echo:
  CHAN OF ANY screen IS echo: -- echo channel with scope local to this PAR only
 PAR
        ------------------------------------------------------------
 SEQ
      keystream.from.file (from.user.filer[2], to.user.filer[2],
              keyboard,1, input.error)
    -- check input.error when real screen accessible again
        ------------------------------------------------------------
   scrstream.sink (screen)    -- consume everything echoed
        ------------------------------------------------------------
  INT x:
  INT kchar:
 SEQ
  j := 0
   {{{ read a sequence of real numbers
  kchar := 0
  x := 1(INT)
   write.char(screen, '$')
   WHILE (x <> 1000000(INT)) AND
      (kchar <> ft.terminated)
  SEQ
     write.char(screen, '>')
     read.echo.char (keyboard, screen, kchar)
   IF
     kchar < 0
     SKIP
      kchar = (INT'#')
       INT hexx RETYPES x:
        read.echo.hex.int (keyboard, screen, hexx, kchar)
    TRUE
       read.echo.int (keyboard, screen, x, kchar)
   IF
     kchar = ft.terminated
     SKIP
    TRUE
    SEQ
     IF
        kchar = ft.number.error
        beep (screen)
      TRUE
       SKIP
      ax[j] := x
      j := j + 1
 }}}
  newline (screen)
  {{{ consume rest of file if any
 IF
    (kchar >= 0) OR (kchar = ft.number.error)
     keystream.sink (keyboard) -- consume the rest of the keyboard file
```

```
      TRUE
        SKIP  -- keyboard file has terminated or failed
    }}}
      write.endstream (screen) -- terminate scrstream.sink
        ------------------------------------------------------------
  {{{  test input.error, if OK tabulate
  IF
    {{{  input error
    input.error <> 0
    SEQ
        write.full.string (screen, "File reading error: ")
      write.int (screen, input.error, 0)
      newline (screen)
   }}}
   TRUE
    SKIP
  }}}
 }}}
 SEQ i=0 FOR M
  app.in ! ax[i]
}}}
{{{  write
[3]BYTE str:
[3]BYTE str1:
[3]BYTE str2:
[128][128]INT outarray:
 INT len,result:
VAL attr IS [ft.opstext, fc.source.text, 2]:
 INT fold.member:
 CHAN OF ANY from.ws IS from.user.filer[0]:
 CHAN OF ANY to.ws IS to.user.filer[0]:
SEQ
  {{{  vals
  VAL pi IS 3.1415 (REAL32):
  VAL pi2 IS 9.8596 (REAL32):
  VAL M IS 3900:
  VAL N IS 64:
  VAL b1 IS 0.5(REAL32):
  VAL b2 IS 2.0(REAL32):
  VAL b3 IS 1.0(REAL32):
  VAL b4 IS 0.0(REAL32):
  VAL b5 IS 255.0(REAL32):
  VAL X IS 65:--number of rings
  VAL rays IS 65:
  VAL Y IS 120:--number of sectors
  VAL views IS 60:
 }}}
 SEQ j=0 FOR 2*N
  SEQ i=0 FOR 2*N
   app.out ? outarray[j][i]

SEQ
  len:=0
  append.int (len, str1, 2*N, 0)
  len:=0
  append.int (len, str2, 2*N, 0)
   create.new.fold(from.ws, to.ws,
       fold.member,"xydata", attr, "", result)
```

```
      write.record.item (from.ws, to.ws, str1, result)
      write.record.item (from.ws, to.ws, str2, result)
    SEQ j=0 FOR 2*N
     SEQ i=0 FOR 2*N
     SEQ
       len := 0
        append.int(len, str, outarray[j][i], 3)
        write.record.item (from.ws, to.ws, str, result)
      finish.new.fold(from.ws, to.ws, fold.member, TRUE, result)
  }}}
 }}}
:

{{{  running
INT any:
SEQ
  write.full.string (screen, "started running--")
  newline(screen)
}}}
{{{  read & write
 read.write()
}}}
{{{  finished and pause
   write.full.string(screen,"finished  running--")
 keyboard ? any
}}}
```

2. This is the network program for convolution, backprojection and interpolation.

```
{{{  SC conl0

#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC conl0(CHAN OF ANY input, output, receive, send)
 [120][65]INT F:
 [3900]INT conl:
 SEQ
   [7800]INT V RETYPES F:
   {{{  convolution
   [3900]INT ax:
   [7800]REAL32 h:
   INT k,Mp:
   REAL32 w0:
   {{{  VALS
   VAL a1 IS 1.0(REAL32):
   VAL a2 IS 4.0(REAL32):
   VAL a3 IS 0.0(REAL32):
   VAL pi2 IS 9.8596 (REAL32):
   VAL M IS 3900:
   VAL P IS 17:
   VAL t IS 1.0(REAL32):
   }}}
   SEQ
     Mp:=M/P
     SEQ n=0 FOR ((2*M)-1)
     SEQ
       k:=(n-(M-1))/2
      IF
        n=(M-1)
         h[n]:=a1/a2
         ((n-(M-1))-(2*k))=0
         h[n]:=a3
          ((n-(M-1))-(2*k))<>0
        SEQ
           w0:= (REAL32 ROUND (n-(M-1)))
           h[n]:=-(a1/(w0*(w0*(pi2))))

    SEQ r=0 FOR M
     SEQ
       input ? ax[r]
       output ! ax[r]

    output ! (Mp+1)
    SEQ i=0 FOR (Mp+1)
     SEQ
       V[i+(M-1)]:=0
       SEQ j=0 FOR M
         V[i+(M-1)]:=((INT ROUND (h[(i+(M-1))-j]*(REAL32 ROUND          (ax[j]))))+V[i+(M-
1)])
```

```
      conl[i]:=V[i+(M-1)]
     output ! conl[i]

  SEQ i=0 FOR M
    receive ? conl[i]
}}}
{{{  backprojection
[3965]INT vol:
[120][1]INT m,H,E,G,Q:
INT a,b,ZZ:
REAL32 r1,o,z,z1,i1,X1,j1,Y1,rays1:
{{{  VALS
VAL M IS 3900:
VAL X IS 65:--number of rings
VAL rays IS 65:
VAL Y IS 120:--number of sectors
VAL views IS 60:
VAL B IS 1:
VAL pi IS 3.1415926(REAL32):
VAL c3 IS 0.5(REAL32):
VAL c2 IS 2.0(REAL32):
VAL c1 IS 1.0(REAL32):
}}}
SEQ
 output ! B
  {{{  initialize
 SEQ
   SEQ j=0 FOR Y
    SEQ i=0 FOR B
     SEQ
       F[j][i] := 0
        a := (5*rays)-(j+i)
       b := a/rays
        m[j][i] := a-(b*rays)
       Q[j][i] := 0
        i1 := (REAL32 ROUND i)
        X1 := (REAL32 ROUND X)
        r1 := (i1+c3)/X1
        j1 := (REAL32 ROUND j)
        Y1 := (REAL32 ROUND Y)
         o := ((c2*pi)*((j1+c3)/Y1))
        z := (r1*COS(o))+c1
        rays1 := (REAL32 ROUND rays)
        z1 := z/(c2/rays1)
        ZZ := (INT TRUNC z1)

      IF
       j>0
         E[j-1][i] := ZZ
       TRUE
         E[Y-1][i] := ZZ
       G[j][i] := ZZ
}}}

 SEQ r=0 FOR (M+rays)
 SEQ
  IF
    r<M
```

```
        vol[r]:=con[r]
      TRUE
        vol[r]:=0
    output ! Q[0][0]
    {{{ shiftQ
    SEQ
      SEQ j=1 FOR (Y-1)
       SEQ i=0 FOR B
         H[j][i] := Q[j-1][i]
      H[0][0]:=vol[r]

      SEQ j=0 FOR Y
       SEQ i=0 FOR B
         Q[j][i]:=H[j][i]
    }}}
    {{{ checkG
    SEQ
      SEQ j=0 FOR Y
       SEQ i=0 FOR B
       SEQ
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
         TRUE
          SKIP
    }}}
    {{{ incM
    SEQ
      SEQ j=0 FOR Y
       SEQ i=0 FOR B
       SEQ
          m[j][i]:=m[j][i]+1
        IF
           m[j][i]>(rays-1)
          SEQ
            m[j][i]:=0
            E[j][i]:=G[j][i]
           IF
            j=0
               G[j][i]:=E[Y-1][i]
             TRUE
               G[j][i]:=E[j-1][i]
         TRUE
          SKIP
    }}}

  SEQ j=0 FOR Y
   SEQ
    SEQ i=0 FOR B
     output ! F[j][i]
    SEQ i=0 FOR X
     receive ? F[j][i]
}}}
{{{ interpolation
[128][128]INT f:
 REAL32 min,max,scale:
{{{ vals
VAL Y IS 120:
```

```
   VAL X IS 65:
   VAL N IS 64:
   VAL b5 IS 255.0(REAL32):
   }}}
   SEQ
    SEQ j=0 FOR Y
     SEQ i=0 FOR X
      output ! F[j][i]

    SEQ j=0 FOR 2*N
     SEQ i=0 FOR 2*N
      receive ? f[j][i]

    SEQ
      min:=999999.0(REAL32)
      max:=0.0(REAL32)
      SEQ j=0 FOR 2*N
       SEQ i=0 FOR 2*N
       SEQ
        IF
            min>(REAL32 ROUND (f[j][i]))
             min:=(REAL32 ROUND (f[j][i]))
          TRUE
           SKIP
        IF
            max<(REAL32 ROUND f[j][i])
             max:=(REAL32 ROUND f[j][i])
          TRUE
           SKIP
       scale:=b5/(max-min)
      SEQ j=0 FOR 2*N
       SEQ i=0 FOR 2*N
          send ! (INT ROUND (((REAL32 ROUND f[j][i])-min)*scale))
  }}}
:
}}}F

}}}
{{{ SC conl1
:::A 3 10
{{{F conl1
:::F interf16.tsr
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC conl1(CHAN OF ANY in, out)
 [121][65]INT F:
 SEQ
  [7865]INT V RETYPES F:
  {{{ convolution
  [3900]INT ax,conl:
  [7800]REAL32 h:
  INT k,Mp,i2,i0:
```

```
REAL32 w0:
{{{ VALS
VAL pi2 IS 9.8596 (REAL32):
VAL M IS 3900:
VAL P IS 17:
VAL t IS 1.0(REAL32):
VAL a1 IS 1.0(REAL32):
VAL a2 IS 4.0(REAL32):
VAL a3 IS 0.0(REAL32):
}}}
SEQ
  Mp:=M/P
  SEQ n=0 FOR ((2*M)-1)
  SEQ
     k:=(n-(M-1))/2
   IF
     n=(M-1)
      h[n]:=a1/a2
      ((n-(M-1))-(2*k))=0
      h[n]:=a3
       ((n-(M-1))-(2*k))<>0
     SEQ
        w0:= (REAL32 ROUND (n-(M-1)))
        h[n]:=-(a1/(w0*(w0*(pi2))))

  SEQ r=0 FOR M
  SEQ
    in ? ax[r]
    out ! ax[r]

  in ? i2
  i0:=i2+(Mp+1)
  out ! i0
  SEQ i=i2 FOR (Mp+1)
  SEQ
     V[i+(M-1)]:=0
    SEQ j=0 FOR M
       V[i+(M-1)]:=((INT ROUND (h[(i+(M-1))-j]*
             (REAL32 ROUND (ax[j])))+V[i+(M-1)])
     conl[i]:=V[i+(M-1)]
  SEQ i=0 FOR i2
   in ? conl[i]
  SEQ i=0 FOR i0
   out ! conl[i]
}}}
{{{ backprojection
[120][65]INT m,Q,E,G,H:
 INT a,b,ZZ,u,ir,is:
 REAL32 r1,o,z0,z1,ireal,X1,jreal,Y1,rays1:
{{{ VALS
VAL M IS 3900:
VAL X IS 65:--number of rings
VAL rays IS 65:
VAL Y IS 120:--number of sectors
VAL views IS 60:
VAL A IS 4:
VAL pi IS 3.1415926(REAL32):
VAL c3 IS 0.5(REAL32):
```

```
VAL c2 IS 2.0(REAL32):
VAL c1 IS 1.0(REAL32):
}}}
SEQ
 in ? ir
  is:=(ir+A)
 out ! is
  {{{  initialize
 SEQ
   SEQ j=0 FOR Y
     SEQ i=ir FOR A
     SEQ
       F[j][i] := 0
        a := (5*rays)-(j+i)
       b := a/rays
        m[j][i] := a-(b*rays)
       Q[j][i] := 0
        ireal := (REAL32 ROUND i)
        X1 := (REAL32 ROUND X)
        r1 := (ireal+c3)/X1
        jreal := (REAL32 ROUND j)
        Y1 := (REAL32 ROUND Y)
         o := ((c2*pi)*((jreal+c3)/Y1))
       z0 := (r1*COS(o))+c1
        rays1 := (REAL32 ROUND rays)
        z1 := z0/(c2/rays1)
        ZZ := (INT TRUNC z1)

      IF
       j>0
         E[j-1][i] := ZZ
       TRUE
         E[Y-1][i] := ZZ
       G[j][i] := ZZ
}}}

 SEQ r=0 FOR (M+rays)
 SEQ
   in ? u
    out ! Q[0][(is-1)]
    {{{  shiftQ
  SEQ
     SEQ j=1 FOR (Y-1)
      SEQ i=ir FOR A
        H[j][i] := Q[j-1][i]
     SEQ i=(ir+1) FOR (A-1)
      H[0][i]:=Q[0][i-1]
    H[0][ir]:=u

    SEQ j=0 FOR Y
      SEQ i=ir FOR A
        Q[j][i]:=H[j][i]
  }}}
   {{{  checkG
  SEQ
    SEQ j=0 FOR Y
      SEQ i=ir FOR A
      SEQ
```

```
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
          TRUE
           SKIP
    }}}
    {{{ incM
   SEQ
     SEQ j=0 FOR Y
      SEQ i=ir FOR A
      SEQ
         m[j][i]:=m[j][i]+1
       IF
          m[j][i]>(rays-1)
         SEQ
           m[j][i]:=0
            E[j][i]:=G[j][i]
          IF
           j=0
              G[j][i]:=E[Y-1][i]
            TRUE
              G[j][i]:=E[j-1][i]
         TRUE
          SKIP
   }}}
  SEQ j=0 FOR Y
   SEQ
     SEQ i=0 FOR ir
      in ? F[j][i]
     SEQ i=0 FOR is
      out ! F[j][i]
}}}
{{{ interpolation
REAL32 t1,t2,t0,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15:
[5]REAL32 rd:
[130]REAL32 x,y:
 REAL32 mr,mm,nn,d,z,r,t16,q1,q,Q0,i1,j1,N1:
 INT m1,n1,mk:
[128][128]INT f:
{{{ vals
VAL Y IS 120:
VAL X IS 65:
VAL pi IS 3.1415 (REAL32):
VAL N IS 64:
VAL C IS 8:
VAL b1 IS 0.5(REAL32):
VAL b2 IS 2.0(REAL32):
VAL b3 IS 1.0(REAL32):
VAL b4 IS 0.0(REAL32):
}}}

SEQ
  SEQ j=0 FOR Y
    SEQ i=0 FOR X
      in ? F[j][i]
  SEQ i=0 FOR X
    F[Y][i]:=F[0][i]
```

```
SEQ j=0 FOR (Y+1)
 SEQ i=0 FOR X
  out ! F[j][i]

 d:=b3/(REAL32 ROUND N)
out ! C
SEQ j=0 FOR 2*N
 SEQ i=0 FOR C
  SEQ
    i1:=REAL32 ROUND i
    j1:=REAL32 ROUND j
    N1:=REAL32 ROUND N
    x[i]:=((i1-N1)+b1)*d
    y[j]:=((j1-N1)+b1)*d
    z:=(x[i]*x[i])+(y[j]*y[j])
    r:=SQRT(z)
    t16:=(y[j])/(x[i])
    q1:=ATAN(t16)
   IF
    y[j]>=b4
     IF
      x[i]>=b4
       q:=q1
      x[i]<b4
       q:=pi+q1
    y[j]<b4
     IF
      x[i]>=b4
        q:=(b2*pi)+q1
      x[i]<b4
       q:=pi+q1
   IF
    r>b3
     f[j][i]:=0
    r<=b3
      {{{ cal nearest point
     SEQ
        Q0:=(b2*pi)/(REAL32 ROUND Y)
       mm:=q/Q0
        m1:=INT TRUNC mm
       nn:=r/d
        n1:=INT TRUNC nn
      SEQ
              ------------------------------------------------------------
       SEQ
            t1:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
           t15:=ABS(t1-((REAL32 ROUND n1)*d))
            t2:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
           t0:=POWER(t15,b2)+POWER(t2,b2)
          rd[1]:=SQRT(t0)
              ------------------------------------------------------------
       SEQ
            t3:=(r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
           t13:=ABS(t3-((REAL32 ROUND n1)*d))
            t4:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
           t5:=POWER(t13,b2)+POWER(t4,b2)
          rd[2]:=SQRT(t5)
              ------------------------------------------------------------
```

```
            SEQ
                t6:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
                t14:=ABS(t6-((REAL32 ROUND (n1+1))*d))
                t7:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
               t8:=POWER(t14,b2)+POWER(t7,b2)
              rd[3]:=SQRT(t8)
                    ------------------------------------------------------------
            SEQ
                t9:=r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0)))
                t12:=ABS(t9-((REAL32 ROUND (n1+1))*d))
                t10:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
               t11:=POWER(t12,b2)+POWER(t10,b2)
              rd[4]:=SQRT(t11)
                    ------------------------------------------------------------
             {{{ decide the shortest rd
            SEQ
              mk:=0
               mr:=1.0(REAL32)
              SEQ k=1 FOR 4
              SEQ
               IF
                  mr>=rd[k]
                 SEQ
                    mr:=rd[k]
                    mk:=k
                TRUE
                 SKIP
               IF
                 mk=1
                    f[j][i]:=F[m1][n1]
                 mk=2
                    f[j][i]:=F[m1+1][n1]
                 mk=3
                    f[j][i]:=F[m1][n1+1]
                 mk=4
                    f[j][i]:=F[m1+1][n1+1]
                 TRUE
                  SKIP
             }}}
           }}}
         out ! f[j][i]
    }}}
:
}}} F

}}}
{{{ SC conl2.15
:::A 3 10
{{{F conl2.15
:::F interf18.tsr
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath
```

```
PROC conl2.15(CHAN OF ANY in, out, VAL INT s)
 [121][65]INT F:
 SEQ
  [7865]INT V RETYPES F:
  {{{ convolution
  [3900]INT ax,conl:
  [7800]REAL32 h:
   INT k,Mp,M.Its,i2,i0:
  REAL32 w0:
  {{{ VALS
  VAL pi2 IS 9.8596 (REAL32):
  VAL M IS 3900:
  VAL P IS 17:
  VAL t IS 1.0(REAL32):
  VAL a1 IS 1.0(REAL32):
  VAL a2 IS 4.0(REAL32):
  VAL a3 IS 0.0(REAL32):
  }}}
  SEQ
   Mp:=M/P
   IF
    {{{ first M REM P transputers
    s<7
     SEQ
      M.Its:=Mp+1
       SEQ n=0 FOR ((2*M)-1)
       SEQ
          k:=(n-(M-1))/2
        IF
          n=(M-1)
           h[n]:=a1/a2
           ((n-(M-1))-(2*k))=0
           h[n]:=a3
           ((n-(M-1))-(2*k))<>0
          SEQ
              w0:= (REAL32 ROUND (n-(M-1)))
              h[n]:=-(a1/(w0*(w0*(pi2))))

      SEQ r=0 FOR M
      SEQ
        in ? ax[r]
        out ! ax[r]

      in ? i2
       i0:=i2+M.Its
      out ! i0
       SEQ i=i2 FOR M.Its
       SEQ
          V[i+(M-1)]:=0
         SEQ j=0 FOR M
            V[i+(M-1)]:=((INT ROUND (h[(i+(M-1))-j]*
                  (REAL32 ROUND (ax[j]))))+V[i+(M-1)])
        conl[i]:=V[i+(M-1)]
      SEQ i=0 FOR i2
        in ? conl[i]
      SEQ i=0 FOR i0
       out ! conl[i]
  }}}
```

```
  {{{ the others
  TRUE
  SEQ
    M.Its:=Mp
    SEQ n=0 FOR ((2*M)-1)
    SEQ
       k:=(n-(M-1))/2
     IF
       n=(M-1)
        h[n]:=a1/a2
        ((n-(M-1))-(2*k))=0
        h[n]:=a3
        ((n-(M-1))-(2*k))<>0
       SEQ
          w0:= (REAL32 ROUND (n-(M-1)))
          h[n]:=-(a1/(w0*(w0*(pi2))))

    SEQ r=0 FOR M
    SEQ
      in ? ax[r]
      out ! ax[r]

    in ? i2
    i0:=i2+M.Its
    out ! i0
    SEQ i=i2 FOR M.Its
    SEQ
       V[i+(M-1)]:=0
      SEQ j=0 FOR M
         V[i+(M-1)]:=((INT ROUND (h[(i+(M-1))-j]*
                (REAL32 ROUND (ax[j]))))+V[i+(M-1)])
       conl[i]:=V[i+(M-1)]
    SEQ i=0 FOR i2
     in ? conl[i]
    SEQ i=0 FOR i0
     out ! conl[i]
  }}}
}}}
{{{ backprojection
[120][65]INT m,Q,E,G,H:
INT a,b,ZZ,u,ir,is:
REAL32 r1,o,z0,z1,ireal,X1,jreal,Y1,rays1:
{{{ VALS
VAL M IS 3900:
VAL X IS 65:--number of rings
VAL rays IS 65:
VAL Y IS 120:--number of sectors
VAL views IS 60:
VAL A IS 4:
VAL c3 IS 0.5(REAL32):
VAL c2 IS 2.0(REAL32):
VAL c1 IS 1.0(REAL32):
VAL pi IS 3.1415926(REAL32):
}}}
SEQ
 in ? ir
 is:=(ir+A)
 out ! is
```

```
{{{ initialize
SEQ
  SEQ j=0 FOR Y
    SEQ i=ir FOR A
    SEQ
      F[j][i] := 0
       a := (5*rays)-(j+i)
      b := a/rays
       m[j][i] := a-(b*rays)
      Q[j][i] := 0
       ireal := (REAL32 ROUND i)
       X1 := (REAL32 ROUND X)
       r1 := (ireal+c3)/X1
       jreal := (REAL32 ROUND j)
       Y1 := (REAL32 ROUND Y)
        o := ((c2*pi)*((jreal+c3)/Y1))
       z0 := (r1*COS(o))+c1
       rays1 := (REAL32 ROUND rays)
       z1 := z0/(c2/rays1)
       ZZ := (INT TRUNC z1)

     IF
       j>0
         E[j-1][i] := ZZ
       TRUE
         E[Y-1][i] := ZZ
       G[j][i] := ZZ
}}}

SEQ r=0 FOR (M+rays)
 SEQ
  in ? u
   out ! Q[0][(is-1)]
   {{{ shiftQ
  SEQ
    SEQ j=1 FOR (Y-1)
     SEQ i=ir FOR A
       H[j][i] := Q[j-1][i]
     SEQ i=(ir+1) FOR (A-1)
      H[0][i]:=Q[0][i-1]
     H[0][ir]:=u

     SEQ j=0 FOR Y
      SEQ i=ir FOR A
        Q[j][i]:=H[j][i]
  }}}
   {{{ checkG
  SEQ
    SEQ j=0 FOR Y
     SEQ i=ir FOR A
     SEQ
      IF
         G[j][i]=m[j][i]
           F[j][i]:=F[j][i]+Q[j][i]
        TRUE
         SKIP
  }}}
   {{{ incM
```

```
    SEQ
      SEQ j=0 FOR Y
       SEQ i=ir FOR A
       SEQ
          m[j][i]:=m[j][i]+1
        IF
           m[j][i]>(rays-1)
          SEQ
            m[j][i]:=0
            E[j][i]:=G[j][i]
           IF
            j=0
               G[j][i]:=E[Y-1][i]
            TRUE
               G[j][i]:=E[j-1][i]
         TRUE
          SKIP
    }}}
  SEQ j=0 FOR Y
   SEQ
     SEQ i=0 FOR ir
      in ? F[j][i]
     SEQ i=0 FOR is
      out ! F[j][i]
}}}
{{{ interpolation
[128][128]INT f:
REAL32 t1,t2,t0,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15:
[5]REAL32 rd:
[130]REAL32 x,y:
 REAL32 mr,mm,nn,d,z,r,t16,q1,q,Q0,i1,j1,N1:
 INT m1,n1,mk:
 INT ised,irec:
{{{ vals
VAL Y IS 120:
VAL X IS 65:
VAL pi IS 3.1415 (REAL32):
VAL N IS 64:
VAL C IS 8:
VAL b1 IS 0.5(REAL32):
VAL b2 IS 2.0(REAL32):
VAL b3 IS 1.0(REAL32):
VAL b4 IS 0.0(REAL32):
VAL b5 IS 255.0(REAL32):
}}}

SEQ
  SEQ j=0 FOR (Y+1)
   SEQ i=0 FOR X
   SEQ
     in ? F[j][i]
     out ! F[j][i]

  d:=b3/(REAL32 ROUND N)
  in ? irec
  ised:=irec+C
  out ! ised
  SEQ j=0 FOR 2*N
```

```
SEQ i=irec FOR C
SEQ
  i1:=REAL32 ROUND i
  j1:=REAL32 ROUND j
  N1:=REAL32 ROUND N
  x[i]:=((i1-N1)+b1)*d
  y[j]:=((j1-N1)+b1)*d
  z:=(x[i]*x[i])+(y[j]*y[j])
  r:=SQRT(z)
  t16:=(y[j])/(x[i])
  q1:=ATAN(t16)
 IF
  y[j]>=b4
   IF
    x[i]>=b4
     q:=q1
    x[i]<b4
     q:=pi+q1
  y[j]<b4
   IF
    x[i]>=b4
      q:=(b2*pi)+q1
    x[i]<b4
     q:=pi+q1
 IF
  r>b3
   f[j][i]:=0
  r<=b3
    {{{ cal nearest point
  SEQ
     Q0:=(b2*pi)/(REAL32 ROUND Y)
    mm:=q/Q0
     m1:=INT TRUNC mm
    nn:=r/d
     n1:=INT TRUNC nn
   SEQ
            ------------------------------------------------------------
    SEQ
        t1:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
       t15:=ABS(t1-((REAL32 ROUND n1)*d))
        t2:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
       t0:=POWER(t15,b2)+POWER(t2,b2)
      rd[1]:=SQRT(t0)
            ------------------------------------------------------------
    SEQ
        t3:=(r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
       t13:=ABS(t3-((REAL32 ROUND n1)*d))
        t4:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
       t5:=POWER(t13,b2)+POWER(t4,b2)
      rd[2]:=SQRT(t5)
            ------------------------------------------------------------
    SEQ
        t6:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
       t14:=ABS(t6-((REAL32 ROUND (n1+1))*d))
        t7:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
       t8:=POWER(t14,b2)+POWER(t7,b2)
      rd[3]:=SQRT(t8)
            ------------------------------------------------------------
```

```
            SEQ
                t9:=r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0)))
                t12:=ABS(t9-((REAL32 ROUND (n1+1))*d))
                 t10:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
                t11:=POWER(t12,b2)+POWER(t10,b2)
               rd[4]:=SQRT(t11)
                      ------------------------------------------------------------
              {{{  decide the shortest rd
            SEQ
              mk:=0
               mr:=1.0(REAL32)
              SEQ k=1 FOR 4
              SEQ
               IF
                  mr>=rd[k]
                 SEQ
                    mr:=rd[k]
                    mk:=k
                TRUE
                 SKIP
              IF
                mk=1
                   f[j][i]:=F[m1][n1]
                mk=2
                   f[j][i]:=F[m1+1][n1]
                mk=3
                   f[j][i]:=F[m1][n1+1]
                mk=4
                   f[j][i]:=F[m1+1][n1+1]
                TRUE
                 SKIP
            }}}
          }}}
     SEQ j=0 FOR 2*N
     SEQ
       SEQ i=0 FOR irec
        in ? f[j][i]
       SEQ i=0 FOR ised
        out ! f[j][i]
   }}}
:
}}} F

}}}
{{{  SC conl16
:::A  3 10
{{{F conl16
:::F  interf17.tsr
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC conl16(CHAN OF ANY in, out)
 [121][65]INT F:
```

```
SEQ
 [7865]INT V RETYPES F:
 {{{ convolution
 [3900]INT ax,conl:
 [7800]REAL32 h:
 INT Mp,k,i2:
 REAL32 w0:
 {{{ VALS
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL P IS 17:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 }}}
 SEQ
  Mp:=M/P
  SEQ n=0 FOR ((2*M)-1)
  SEQ
    k:=(n-(M-1))/2
   IF
     n=(M-1)
      h[n]:=a1/a2
      ((n-(M-1))-(2*k))=0
      h[n]:=a3
      ((n-(M-1))-(2*k))<>0
     SEQ
        w0:= (REAL32 ROUND (n-(M-1)))
        h[n]:=-(a1/(w0*(w0*(pi2))))

  SEQ r=0 FOR M
   in ? ax[r]

  in ? i2
  SEQ i=i2 FOR Mp
  SEQ
    V[i+(M-1)]:=0
   SEQ j=0 FOR M
      V[i+(M-1)]:=((INT ROUND (h[(i+(M-1))-j]*
            (REAL32 ROUND (ax[j]))))+V[i+(M-1)])
    conl[i]:=V[i+(M-1)]
  SEQ i=0 FOR i2
   in ? conl[i]
  SEQ i=0 FOR (i2+Mp)
   out ! conl[i]
 }}}
 {{{ backprojection
 [120][65]INT m,Q,H,E,G:
 INT a,b,ZZ,u,ir:
 REAL32 r1,o,z0,z1,ireal,X1,jreal,Y1,rays1:
 {{{ VALS
 VAL M IS 3900:
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 VAL A IS 4:
```

```
VAL c3 IS 0.5(REAL32):
VAL c2 IS 2.0(REAL32):
VAL c1 IS 1.0(REAL32):
VAL pi IS 3.1415926(REAL32):
}}}
SEQ
 in ? ir
 {{{ initialize
SEQ
  SEQ j=0 FOR Y
   SEQ i=ir FOR A
   SEQ
     F[j][i] := 0
      a := (5*rays)-(j+i)
     b := a/rays
      m[j][i] := a-(b*rays)
     Q[j][i] := 0
      ireal := (REAL32 ROUND i)
      X1 := (REAL32 ROUND X)
      r1 := (ireal+c3)/X1
      jreal := (REAL32 ROUND j)
      Y1 := (REAL32 ROUND Y)
       o := ((c2*pi)*((jreal+c3)/Y1))
     z0 := (r1*COS(o))+c1
      rays1 := (REAL32 ROUND rays)
      z1 := z0/(c2/rays1)
      ZZ := (INT TRUNC z1)

    IF
     j>0
       E[j-1][i] := ZZ
     TRUE
       E[Y-1][i] := ZZ
     G[j][i] := ZZ
}}}

SEQ r=0 FOR (M+rays)
SEQ
  in ? u
  {{{ shiftQ
 SEQ
    SEQ j=1 FOR (Y-1)
     SEQ i=ir FOR A
       H[j][i] := Q[j-1][i]
    SEQ i=(ir+1) FOR (A-1)
     H[0][i]:=Q[0][i-1]
   H[0][ir]:=u

   SEQ j=0 FOR Y
    SEQ i=ir FOR A
      Q[j][i]:=H[j][i]
 }}}
  {{{ checkG
 SEQ
   SEQ j=0 FOR Y
    SEQ i=ir FOR A
    SEQ
     IF
```

```
            G[j][i]=m[j][i]
              F[j][i]:=F[j][i]+Q[j][i]
            TRUE
             SKIP
      }}}
      {{{ incM
      SEQ
        SEQ j=0 FOR Y
          SEQ i=ir FOR A
          SEQ
             m[j][i]:=m[j][i]+1
           IF
              m[j][i]>(rays-1)
             SEQ
               m[j][i]:=0
               E[j][i]:=G[j][i]
              IF
               j=0
                  G[j][i]:=E[Y-1][i]
                TRUE
                  G[j][i]:=E[j-1][i]
            TRUE
             SKIP
      }}}
    SEQ j=0 FOR Y
     SEQ
       SEQ i=0 FOR ir
        in ? F[j][i]
       SEQ i=0 FOR (ir+A)
        out ! F[j][i]
}}}
{{{ interpolation
[128][128]INT f:
REAL32 t1,t2,t0,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15:
[5]REAL32 rd:
[130]REAL32 x,y:
 REAL32 mr,mm,nn,d,z,r,t16,q1,q,Q0,i1,j1,N1:
 INT m1,n1,mk,ised,irec:
{{{ vals
VAL Y IS 120:
VAL X IS 65:
VAL pi IS 3.1415 (REAL32):
VAL N IS 64:
VAL C IS 8:
VAL b1 IS 0.5(REAL32):
VAL b2 IS 2.0(REAL32):
VAL b3 IS 1.0(REAL32):
VAL b4 IS 0.0(REAL32):
}}}

SEQ
  SEQ j=0 FOR (Y+1)
   SEQ i=0 FOR X
     in ? F[j][i]

  d:=b3/(REAL32 ROUND N)
  in ? irec
  ised:=irec+C
```

```
SEQ j=0 FOR 2*N
 SEQ i=irec FOR C
 SEQ
   i1:=REAL32 ROUND i
   j1:=REAL32 ROUND j
   N1:=REAL32 ROUND N
   x[i]:=((i1-N1)+b1)*d
   y[j]:=((j1-N1)+b1)*d
    z:=(x[i]*x[i])+(y[j]*y[j])
   r:=SQRT(z)
   t16:=(y[j])/(x[i])
   q1:=ATAN(t16)
  IF
   y[j]>=b4
    IF
     x[i]>=b4
      q:=q1
     x[i]<b4
      q:=pi+q1
   y[j]<b4
    IF
     x[i]>=b4
       q:=(b2*pi)+q1
     x[i]<b4
      q:=pi+q1
  IF
   r>b3
    f[j][i]:=0
   r<=b3
    {{{ cal nearest point
   SEQ
      Q0:=(b2*pi)/(REAL32 ROUND Y)
     mm:=q/Q0
      m1:=INT TRUNC mm
     nn:=r/d
      n1:=INT TRUNC nn
    SEQ
            ------------------------------------------------------------
     SEQ
        t1:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
       t15:=ABS(t1-((REAL32 ROUND n1)*d))
        t2:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
       t0:=POWER(t15,b2)+POWER(t2,b2)
      rd[1]:=SQRT(t0)
            ------------------------------------------------------------
     SEQ
        t3:=(r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
       t13:=ABS(t3-((REAL32 ROUND n1)*d))
        t4:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
       t5:=POWER(t13,b2)+POWER(t4,b2)
      rd[2]:=SQRT(t5)
            ------------------------------------------------------------
     SEQ
        t6:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
       t14:=ABS(t6-((REAL32 ROUND (n1+1))*d))
        t7:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
       t8:=POWER(t14,b2)+POWER(t7,b2)
      rd[3]:=SQRT(t8)
```

```
                    ----------------------------------------------------------
            SEQ
                t9:=r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0)))
                t12:=ABS(t9-((REAL32 ROUND (n1+1))*d))
                t10:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
                t11:=POWER(t12,b2)+POWER(t10,b2)
                rd[4]:=SQRT(t11)
                    ----------------------------------------------------------
              {{{ decide the shortest rd
            SEQ
              mk:=0
               mr:=1.0(REAL32)
              SEQ k=1 FOR 4
              SEQ
               IF
                  mr>=rd[k]
                 SEQ
                    mr:=rd[k]
                    mk:=k
                TRUE
                 SKIP
              IF
                mk=1
                   f[j][i]:=F[m1][n1]
                mk=2
                   f[j][i]:=F[m1+1][n1]
                mk=3
                   f[j][i]:=F[m1][n1+1]
                mk=4
                   f[j][i]:=F[m1+1][n1+1]
                TRUE
                 SKIP
            }}}
          }}}
      SEQ j=0 FOR 2*N
       SEQ
         SEQ i=0 FOR irec
          in ? f[j][i]
         SEQ i=0 FOR ised
          out ! f[j][i]
    }}}
:
}}} F
}}}
{{{  configuration
{{{  link constants
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:
}}}
{{{  vals
VAL P IS 17:
}}}
```

```
{{{  CHANs
[P]CHAN OF ANY pipe:
CHAN OF ANY app.in, app.out:
}}}

PLACED PAR
 PROCESSOR 0 T4
   PLACE app.in AT link0in:
   PLACE app.out AT link0out:
   PLACE pipe[0] AT link1out:
   PLACE pipe[P-1] AT link3in:
   conl0 ( app.in, pipe[0], pipe[P-1], app.out)

 PROCESSOR 1 T4
   PLACE pipe[0] AT link0in:
   PLACE pipe[1] AT link1out:
   conl1 (pipe[0], pipe[1])

 PLACED PAR s=2 FOR (P-3)
  PROCESSOR s T4
    PLACE pipe[s-1] AT link0in:
    PLACE pipe[s] AT link1out:
    conl2.15 ( pipe[s-1], pipe[s], s)

 PROCESSOR (P-1) T4
   PLACE pipe[P-2] AT link0in:
   PLACE pipe[P-1] AT link2out:
   conl16 ( pipe[P-2], pipe[P-1] )
}}}
```

3. This is the exe program for convolution.

```
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC read.write()
 {{{ hard channel placement values
 VAL link0out IS 0:
 VAL link1out IS 1:
 VAL link2out IS 2:
 VAL link3out IS 3:
 VAL link0in IS 4:
 VAL link1in IS 5:
 VAL link2in IS 6:
 VAL link3in IS 7:
 }}}
 {{{ channel
 CHAN OF ANY app.in, app.out:
 }}}
 {{{ placements
 PLACE app.in AT link2out:
 PLACE app.out AT link2in:
 }}}
 {{{ vals
 VAL pi IS 3.1415 (REAL32):
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 VAL N IS 64:
 VAL b1 IS 0.5(REAL32):
 VAL b2 IS 2.0(REAL32):
 VAL b3 IS 1.0(REAL32):
 VAL b4 IS 0.0(REAL32):
 VAL b5 IS 255.0(REAL32):
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}
 {{{ main program
 PAR
  {{{ read
  [4000]INT ax:
  INT k,j:
  SEQ
    {{{ vals
    VAL M IS 3900:
    }}}
    {{{ read original data in
```

```
 INT input.error:
SEQ
  CHAN OF INT filekeys:
   CHAN OF INT keyboard IS filekeys: -- channel from simulated keyboard
  CHAN OF ANY echo:
  CHAN OF ANY screen IS echo: -- echo channel with scope local to this PAR only
  PAR
          ------------------------------------------------------------
  SEQ
       keystream.from.file (from.user.filer[2], to.user.filer[2],
                 keyboard,1, input.error)
     -- check input.error when real screen accessible again
          ------------------------------------------------------------
    scrstream.sink (screen)    -- consume everything echoed
          ------------------------------------------------------------
   INT x:
   INT kchar:
  SEQ
   j := 0
    {{{ read a sequence of real numbers
   kchar := 0
   x := 1(INT)
    write.char(screen, '$')
   WHILE (x <> 1000000(INT)) AND
       (kchar <> ft.terminated)
   SEQ
      write.char(screen, '>')
      read.echo.char (keyboard, screen, kchar)
    IF
      kchar < 0
      SKIP
       kchar = (INT'#')
        INT hexx RETYPES x:
         read.echo.hex.int (keyboard, screen, hexx, kchar)
     TRUE
        read.echo.int (keyboard, screen, x, kchar)
    IF
      kchar = ft.terminated
      SKIP
     TRUE
     SEQ
      IF
         kchar = ft.number.error
         beep (screen)
       TRUE
        SKIP
       ax[j] := x
       j := j + 1
 }}}
   newline (screen)
   {{{ consume rest of file if any
 IF
    (kchar >= 0) OR (kchar = ft.number.error)
      keystream.sink (keyboard) -- consume the rest of the keyboard file
   TRUE
     SKIP  -- keyboard file has terminated or failed
 }}}
    write.endstream (screen) -- terminate scrstream.sink
```

```
      ----------------------------------------------------------
      {{{ test input.error, if OK tabulate
      IF
        {{{ input error
        input.error <> 0
        SEQ
            write.full.string (screen, "File reading error: ")
          write.int (screen, input.error, 0)
          newline (screen)
        }}}
        TRUE
         SKIP
      }}}
     }}}
     SEQ i=0 FOR M
       app.in ! ax[i]
   }}}
   {{{ write
   [8]BYTE str:
   [4000]INT conl:
   INT len,result:
   VAL attr IS [ft.opstext, fc.source.text, 2]:
    INT fold.member:
    CHAN OF ANY from.ws IS from.user.filer[0]:
    CHAN OF ANY to.ws IS to.user.filer[0]:
   SEQ
     {{{ vals
     VAL pi IS 3.1415 (REAL32):
     VAL pi2 IS 9.8596 (REAL32):
     VAL M IS 3900:
     VAL N IS 64:
     VAL b1 IS 0.5(REAL32):
     VAL b2 IS 2.0(REAL32):
     VAL b3 IS 1.0(REAL32):
     VAL b4 IS 0.0(REAL32):
     VAL b5 IS 255.0(REAL32):
     VAL X IS 65:--number of rings
     VAL rays IS 65:
     VAL Y IS 120:--number of sectors
     VAL views IS 60:
     }}}
     SEQ i=0 FOR M
       app.out ? conl[i]

    SEQ
       create.new.fold(from.ws, to.ws,
            fold.member,"xydata", attr, "", result)
      SEQ i=0 FOR M
      SEQ
        len := 0
         append.int(len, str, conl[i], 8)
          write.record.item (from.ws, to.ws, str, result)
        finish.new.fold(from.ws, to.ws, fold.member, TRUE, result)
  }}}
 }}}
:

{{{  running
```

```
INT any:
SEQ
  write.full.string (screen, "started running--")
  newline(screen)
}}}
{{{ read & write
  read.write()
}}}
{{{ finished and pause
   write.full.string(screen,"finished  running--")
  keyboard ? any
}}}
```

4. This is the target program for convolution only.

```
{{{ SC conl
{{{F conl
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC conl(CHAN OF ANY input, output)
 [4000]INT ax,conl:
 [8000]INT k:
 [8000]REAL32 h,Q,w:
 {{{ vals
 VAL pi IS 3.1415 (REAL32):
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 VAL N IS 64:
 VAL b1 IS 0.5(REAL32):
 VAL b2 IS 2.0(REAL32):
 VAL b3 IS 1.0(REAL32):
 VAL b4 IS 0.0(REAL32):
 VAL b5 IS 255.0(REAL32):
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}

 SEQ
  SEQ n=0 FOR ((2*M)-1)
   SEQ
     k[n]:=(n-(M-1))/2
    IF
      n=(M-1)
       h[n]:=a1/a2
       ((n-(M-1))-(2*k[n]))=0
       h[n]:=a3
       ((n-(M-1))-(2*k[n]))<>0
      SEQ
         w[n]:= (REAL32 ROUND (n-(M-1)))
         h[n]:=-(a1/(w[n]*(w[n]*(pi2))))

  SEQ j=0 FOR M
   input ? ax[j]

  SEQ i=0 FOR M
   SEQ
     Q[i+(M-1)]:=a3
     SEQ j=0 FOR M
        Q[i+(M-1)]:=(h[(i+(M-1))-j]*(REAL32 ROUND (ax[j])))+Q[i+(M-1)]
```

```
      conl[i]:=INT ROUND (Q[i+(M-1)])
    output ! conl[i]
:
}}}F

}}}
{{{  configuration
{{{  link constants
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:
}}}
{{{  CHANs
CHAN OF ANY app.in, app.out:
}}}

PLACED PAR
 PROCESSOR 0 T4
   PLACE app.in AT link0in:
   PLACE app.out AT link0out:
   conl ( app.in, app.out)
}}}
```

5. This is the exe program for backprojection only.

```
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC feed.print()
 {{{ hard channel placement values
 VAL link0out IS 0:
 VAL link1out IS 1:
 VAL link2out IS 2:
 VAL link3out IS 3:
 VAL link0in IS 4:
 VAL link1in IS 5:
 VAL link2in IS 6:
 VAL link3in IS 7:
 }}}
 {{{ channel
 CHAN OF ANY app.in, app.out:
 }}}
 {{{ placements
 PLACE app.in AT link2out:
 PLACE app.out AT link2in:
 }}}
 {{{ vals
 VAL M IS 3900:
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}
 {{{ main program
 PAR
  {{{ feed
  [4000]INT ax:
  [4000]INT conl:
  SEQ
    {{{ vals
    VAL M IS 3900:
    VAL views IS 60:
    VAL rays IS 65:
    }}}
    {{{ read convoled data in
    INT j:
    INT input.error:
   SEQ
      CHAN OF INT filekeys:
       CHAN OF INT keyboard IS filekeys: -- channel from simulated keyboard
      CHAN OF ANY echo:
      CHAN OF ANY screen IS echo: -- echo channel with scope local to this PAR only
     PAR
          ----------------------------------------------------------
     SEQ
```

```
      keystream.from.file (from.user.filer[2], to.user.filer[2],
              keyboard,1, input.error)
    -- check input.error when real screen accessible again
      ------------------------------------------------------------
  scrstream.sink (screen)    -- consume everything echoed
      ------------------------------------------------------------
 INT x:
 INT kchar:
SEQ
 j := 0
  {{{ read a sequence of real numbers
 kchar := 0
 x := 1(INT)
  write.char(screen, '$')
 WHILE (x <> 100000) AND
     (kchar <> ft.terminated)
 SEQ
    write.char(screen, '>')
    read.echo.char (keyboard, screen, kchar)
  IF
    kchar < 0
    SKIP
     kchar = (INT'#')
      INT hexx RETYPES x:
       read.echo.hex.int (keyboard, screen, hexx, kchar)
    TRUE
       read.echo.int (keyboard, screen, x, kchar)
  IF
     kchar = ft.terminated
    SKIP
    TRUE
    SEQ
     IF
        kchar = ft.number.error
        beep (screen)
      TRUE
       SKIP
      ax[j] := x
      j := j + 1
 }}}
  newline (screen)
  {{{ consume rest of file if any
IF
   (kchar >= 0) OR (kchar = ft.number.error)
     keystream.sink (keyboard) -- consume the rest of the keyboard file
  TRUE
    SKIP  -- keyboard file has terminated or failed
 }}}
   write.endstream (screen) -- terminate scrstream.sink
      ------------------------------------------------------------
{{{ test input.error, if OK tabulate
IF
  {{{ input error
  input.error <> 0
  SEQ
      write.full.string (screen, "File reading error: ")
    write.int (screen, input.error, 0)
    newline (screen)
```

```
      }}}
      TRUE
       SKIP
     }}}
    }}}
    SEQ i=0 FOR (M+rays)
    SEQ
     IF
       i<M
         conl[i]:=ax[i]
       i>=M
         conl[i]:=0
    SEQ i=0 FOR (M+rays)
     app.in ! conl[i]
  }}}
  {{{ print
  [8]BYTE str:
  [130][70]INT F:
  VAL attr IS [ft.opstext, fc.source.text, 2]:
   INT Y0,X0,q,len,any:
  INT result:
   INT fold.member:
   CHAN OF ANY from.ws IS from.user.filer[0]:
   CHAN OF ANY to.ws IS to.user.filer[0]:
  SEQ
    {{{ vals
    VAL M IS 3900:
    VAL rays IS 65:
    VAL X IS 65:--number of rings
    VAL Y IS 120:--number of sectors
    }}}
    SEQ j=0 FOR Y
     SEQ i=0 FOR X
      app.out ? F[j][i]

   SEQ
      create.new.fold(from.ws, to.ws,
           fold.member,"backdata", attr, "", result)
     SEQ j=0 FOR Y
     SEQ i=0 FOR X
     SEQ
       len := 0
        append.int(len, str, F[j][i],8)
        write.record.item (from.ws, to.ws, str, result)
      finish.new.fold(from.ws, to.ws, fold.member, TRUE, result)
  }}}
 }}}
:

{{{  running
INT  any:
SEQ
  write.full.string (screen, "started running--")
  newline(screen)
}}}
{{{ feed & print
 feed.print()
}}}
```

```
{{{  finished and pause
   write.full.string(screen,"finished  running--")
  keyboard ? any
}}}
```

6. This is the target program for backprojection only.

```
{{{  SC backprojection
{{{F  backprojection
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC backprojection(CHAN OF ANY input, output)
 [130][70]INT F,m,Q,E,G,H:
 [4000]INT conl:
 INT a,b,ZZ,zz,any:
 REAL32 r,o,z,z1,i1,X1,j1,Y1,rays1:
 CHAN OF ANY screen:
 {{{  vals
 VAL X IS 65: --number of rings
 VAL Y IS 120: --number of sectors
 VAL rays IS 65:
 VAL views IS 60:
 VAL M IS 3900:
 VAL pi IS 3.1415926(REAL32):
 VAL a1 IS 0.5(REAL32):
 VAL b1 IS 2.0(REAL32):
 VAL c1 IS 1.0(REAL32):
 }}}

 SEQ
   {{{  initialize
  SEQ
    SEQ j=0 FOR Y
     SEQ i=0 FOR X
     SEQ
       F[j][i] := 0
        a := (5*rays)-(j+i)
       b := a/rays
        m[j][i] := a-(b*rays)
       Q[j][i] := 0
        i1 := (REAL32 ROUND i)
       X1 := (REAL32 ROUND X)
       r := (i1+a1)/X1
       j1 := (REAL32 ROUND j)
       Y1 := (REAL32 ROUND Y)
        o := ((b1*pi)*((j1+a1)/Y1))
       z := (r*COS(o))+c1
        rays1 := (REAL32 ROUND rays)
       z1 := z/(b1/rays1)
       ZZ := (INT TRUNC z1)

      IF
       j>0
         E[j-1][i] := ZZ
       TRUE
         E[Y-1][i] := ZZ
```

```
      G[j][i] := ZZ
  }}}

  SEQ r=0 FOR (M+rays)
   SEQ
    input ? conl[r]
   SEQ
     SEQ j=1 FOR (Y-1)
      SEQ i=0 FOR X
        H[j][i] := Q[j-1][i]
     SEQ i=1 FOR (X-1)
      H[0][i]:=Q[0][i-1]
     H[0][0]:=conl[r]

     SEQ j=0 FOR Y
      SEQ i=0 FOR X
        Q[j][i]:=H[j][i]


    {{{ checkG
    SEQ
     SEQ j=0 FOR Y
      SEQ i=0 FOR X
      SEQ
       IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
         TRUE
          SKIP
    }}}

    {{{ incM
    SEQ
     SEQ j=0 FOR Y
      SEQ i=0 FOR X
      SEQ
         m[j][i]:=m[j][i]+1
       IF
          m[j][i]>(rays-1)
         SEQ
           m[j][i]:=0
           E[j][i]:=G[j][i]
          IF
           j=0
              G[j][i]:=E[Y-1][i]
           TRUE
              G[j][i]:=E[j-1][i]
         TRUE
          SKIP
    }}}


  SEQ j=0 FOR Y
   SEQ i=0 FOR X
    output ! F[j][i]
:
}}} F
```

```
}}}
{{{  configuration
{{{  link constants
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:
}}}
CHAN OF ANY app.in, app.out:

PLACED PAR
 PROCESSOR 0 T4
   PLACE app.in AT link0in:
   PLACE app.out AT link0out:
   backprojection(app.in, app.out)
}}}
```

7. This is the exe program for interpolation only.

```
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC read.send()
 {{{ hard channel placement values
 VAL link0out IS 0:
 VAL link1out IS 1:
 VAL link2out IS 2:
 VAL link3out IS 3:
 VAL link0in IS 4:
 VAL link1in IS 5:
 VAL link2in IS 6:
 VAL link3in IS 7:
 }}}
 {{{ channel
 CHAN OF ANY app.in, app.out:
 }}}
 {{{ placements
 PLACE app.in AT link2out:
 PLACE app.out AT link2in:
 }}}
 {{{ vals
 VAL pi IS 3.1415 (REAL32):
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 VAL N IS 64:
 VAL b1 IS 0.5(REAL32):
 VAL b2 IS 2.0(REAL32):
 VAL b3 IS 1.0(REAL32):
 VAL b4 IS 0.0(REAL32):
 VAL b5 IS 255.0(REAL32):
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}
 {{{ main program
 PAR
  {{{ read
  [10000]INT ax:
   INT k,j,any:
  [130][70]INT F:
  SEQ
    {{{ vals
    VAL M IS 3900:
    VAL Y IS 120:
```

```
VAL X IS 65:
}}}
 {{{  read original data in
 INT input.error:
SEQ
  CHAN OF INT filekeys:
   CHAN OF INT keyboard IS filekeys: -- channel from simulated keyboard
  CHAN OF ANY echo:
  CHAN OF ANY screen IS echo: -- echo channel with scope local to this PAR only
 PAR
        ------------------------------------------------------------
 SEQ
      keystream.from.file (from.user.filer[2], to.user.filer[2],
              keyboard,1, input.error)
    -- check input.error when real screen accessible again
        ------------------------------------------------------------
   scrstream.sink (screen)    -- consume everything echoed
        ------------------------------------------------------------
  INT x:
  INT kchar:
 SEQ
  j := 0
   {{{  read a sequence of real numbers
  kchar := 0
  x := 1(INT)
   write.char(screen, '$')
   WHILE (x <> 1000000(INT)) AND
      (kchar <> ft.terminated)
  SEQ
     write.char(screen, '>')
     read.echo.char (keyboard, screen, kchar)
   IF
     kchar < 0
     SKIP
      kchar = (INT'#')
       INT hexx RETYPES x:
        read.echo.hex.int (keyboard, screen, hexx, kchar)
    TRUE
       read.echo.int (keyboard, screen, x, kchar)
   IF
     kchar = ft.terminated
     SKIP
    TRUE
    SEQ
     IF
        kchar = ft.number.error
        beep (screen)
      TRUE
       SKIP
      ax[j] := x
      j := j + 1
 }}}
  newline (screen)
  {{{  consume rest of file if any
 IF
    (kchar >= 0) OR (kchar = ft.number.error)
      keystream.sink (keyboard) -- consume the rest of the keyboard file
  TRUE
```

```
        SKIP  -- keyboard file has terminated or failed
      }}}
        write.endstream (screen) -- terminate scrstream.sink
          ------------------------------------------------------------
    {{{  test input.error, if OK tabulate
   IF
     {{{  input error
     input.error <> 0
     SEQ
          write.full.string (screen, "File reading error: ")
        write.int (screen, input.error, 0)
        newline (screen)
     }}}
     TRUE
      SKIP
   }}}
  }}}
  SEQ j=0 FOR Y
   SEQ i=0 FOR X
   SEQ
      F[j][i]:=ax[(j*X)+i]
     app.in ! F[j][i]
}}}
{{{  send
[3]BYTE str:
[3]BYTE str1:
[3]BYTE str2:
[130][130]INT outarray:
 INT len,any,result:
VAL attr IS [ft.opstext, fc.source.text, 2]:
 INT fold.member:
 CHAN OF ANY from.ws IS from.user.filer[0]:
 CHAN OF ANY to.ws IS to.user.filer[0]:
SEQ
  {{{  vals
  VAL N IS 64:
  VAL Y IS 120:
  VAL X IS 65:
  }}}
  SEQ j=0 FOR 2*N
   SEQ i=0 FOR 2*N
    app.out ? outarray[j][i]
 SEQ
   len:=0
   append.int(len,str1,2*N,0)
   len:=0
   append.int(len,str2,2*N,0)
    create.new.fold(from.ws, to.ws,
         fold.member,"xydata", attr, "", result)
    write.record.item(from.ws,to.ws,str1,result)
    write.record.item(from.ws,to.ws,str2,result)
   SEQ j=0 FOR 2*N
    SEQ i=0 FOR 2*N
    SEQ
      len := 0
       append.int(len, str, outarray[j][i], 3)
       write.record.item (from.ws, to.ws, str, result)
     finish.new.fold(from.ws, to.ws, fold.member, TRUE, result)
```

```
  }}}
 }}}
:

{{{  running
INT any:
SEQ
  write.full.string (screen, "started running--")
  newline(screen)
}}}
{{{  read & send
 read.send()
}}}
{{{  finished and pause
   write.full.string(screen,"finished  running--")
 keyboard ? any
}}}
```

8. This is the target program for interpolation only.

```
{{{ SC interpolation
{{{F inter
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC inter(CHAN OF ANY input, output)
 [130][70]INT F:
 [130][130]INT outarray:
 [130][130]REAL32 f:
  REAL32 min,max,scale:
 REAL32 t1,t2,t0,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15:
 [5]REAL32 rd:
 [130]REAL32 x,y:
  REAL32 mr,m,n,d,z,r,t16,q1,q,Q0,i1,j1,N1:
  INT m1,n1,mk:
 {{{ vals
 VAL pi IS 3.1415 (REAL32):
 VAL pi2 IS 9.8596 (REAL32):
 VAL M IS 3900:
 VAL t IS 1.0(REAL32):
 VAL a1 IS 1.0(REAL32):
 VAL a2 IS 4.0(REAL32):
 VAL a3 IS 0.0(REAL32):
 VAL N IS 64:
 VAL b1 IS 0.5(REAL32):
 VAL b2 IS 2.0(REAL32):
 VAL b3 IS 1.0(REAL32):
 VAL b4 IS 0.0(REAL32):
 VAL b5 IS 255.0(REAL32):
 VAL X IS 65:--number of rings
 VAL rays IS 65:
 VAL Y IS 120:--number of sectors
 VAL views IS 60:
 }}}
 SEQ
  SEQ j=0 FOR Y
   SEQ i=0 FOR X
    input ? F[j][i]
  SEQ i=0 FOR X
   F[Y][i]:=F[0][i]

  d:=b3/(REAL32 ROUND N)
  SEQ j=0 FOR 2*N
   SEQ i=0 FOR 2*N
    SEQ
      i1:=REAL32 ROUND i
      j1:=REAL32 ROUND j
      N1:=REAL32 ROUND N
      x[i]:=((i1-N1)+b1)*d
      y[j]:=((j1-N1)+b1)*d
       z:=(x[i]*x[i])+(y[j]*y[j])
```

```
 r:=SQRT(z)
 t16:=(y[j])/(x[i])
 q1:=ATAN(t16)
IF
 y[j]>=b4
  IF
   x[i]>=b4
    q:=q1
   x[i]<b4
    q:=pi+q1
 y[j]<b4
  IF
   x[i]>=b4
     q:=(b2*pi)+q1
   x[i]<b4
    q:=pi+q1
IF
 r>b3
  f[j][i]:=b4
 r<=b3
   {{{ cal nearest point
  SEQ
    Q0:=(b2*pi)/(REAL32 ROUND Y)
   m:=q/Q0
    m1:=INT TRUNC m
   n:=r/d
    n1:=INT TRUNC n
   SEQ
          ------------------------------------------------------------
    SEQ
       t1:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
      t15:=ABS(t1-((REAL32 ROUND n1)*d))
       t2:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
      t0:=POWER(t15,b2)+POWER(t2,b2)
     rd[1]:=SQRT(t0)
          ------------------------------------------------------------
    SEQ
       t3:=(r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
      t13:=ABS(t3-((REAL32 ROUND n1)*d))
       t4:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
      t5:=POWER(t13,b2)+POWER(t4,b2)
     rd[2]:=SQRT(t5)
          ------------------------------------------------------------
    SEQ
       t6:=(r*COS(ABS(q-((REAL32 ROUND m1)*Q0))))
      t14:=ABS(t6-((REAL32 ROUND (n1+1))*d))
       t7:=ABS(r*SIN(ABS(q-((REAL32 ROUND m1)*Q0))))
      t8:=POWER(t14,b2)+POWER(t7,b2)
     rd[3]:=SQRT(t8)
          ------------------------------------------------------------
    SEQ
       t9:=r*COS(ABS(q-((REAL32 ROUND (m1+1))*Q0)))
      t12:=ABS(t9-((REAL32 ROUND (n1+1))*d))
       t10:=ABS(r*SIN(ABS(q-((REAL32 ROUND (m1+1))*Q0))))
      t11:=POWER(t12,b2)+POWER(t10,b2)
     rd[4]:=SQRT(t11)
          ------------------------------------------------------------
     {{{ decide the shortest rd
```

```
          SEQ
            mk:=0
             mr:=1.0(REAL32)
            SEQ k=1 FOR 4
            SEQ
             IF
                mr>=rd[k]
               SEQ
                  mr:=rd[k]
                  mk:=k
              TRUE
               SKIP
            IF
              mk=1
                 f[j][i]:=REAL32 ROUND (F[m1][n1])
              mk=2
                 f[j][i]:=REAL32 ROUND (F[m1+1][n1])
              mk=3
                 f[j][i]:=REAL32 ROUND (F[m1][n1+1])
              mk=4
                 f[j][i]:=REAL32 ROUND (F[m1+1][n1+1])
              TRUE
               SKIP
         }}}
       }}}

  SEQ
    min:=999999.0(REAL32)
    max:=0.0(REAL32)
    SEQ j=0 FOR 2*N
     SEQ i=0 FOR 2*N
     SEQ
      IF
        min>f[j][i]
         min:=f[j][i]
       TRUE
       SKIP
      IF
        max<f[j][i]
         max:=f[j][i]
       TRUE
       SKIP
    scale:=b5/(max-min)
   SEQ j=0 FOR 2*N
    SEQ i=0 FOR 2*N
    SEQ
       outarray[j][i]:=INT ROUND ((f[j][i]-min)*scale)
      output ! outarray[j][i]
:
}}}F

}}}
{{{  configuration
{{{  link constants
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
```

```
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:
}}}
{{{  CHANs
CHAN OF ANY app.in, app.out:
}}}

PLACED PAR
 PROCESSOR 0 T4
   PLACE app.in AT link0in:
   PLACE app.out AT link0out:
   inter ( app.in, app.out)
}}}
```

9. This is the network program for backprojection only.

```
{{{ SC interface

{{{F interface
:::F interfac.tsr
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath

PROC interface (CHAN OF ANY input, output,CHAN OF ANY send, receive)
 [120][65]INT F,m,Q,E,G,H:
 [60][65]INT conl:
  INT a,b,ZZ,zz,any,i0,w,u,i2:
  REAL32 r1,o,z,z1,i1,X1,j1,Y1,rays1:
  {{{ vals
 VAL X IS 65:
 VAL B IS 1:
 VAL Y IS 120: --number of sectors
 VAL rays IS 65:
 VAL views IS 60:
 VAL M IS 3900:
 VAL pi IS 3.1415926(REAL32):
 VAL a1 IS 0.5(REAL32):
 VAL b1 IS 2.0(REAL32):
 VAL c1 IS 1.0(REAL32):
 }}}

 SEQ
  output ! B
   {{{ initialize
  SEQ
    SEQ j=0 FOR Y
     SEQ i=0 FOR B
      SEQ
        F[j][i] := 0
         a := (5*rays)-(j+i)
        b := a/rays
         m[j][i] := a-(b*rays)
        Q[j][i] := 0
        i1 := (REAL32 ROUND i)
        X1 := (REAL32 ROUND X)
        r1 := (i1+a1)/X1
        j1 := (REAL32 ROUND j)
        Y1 := (REAL32 ROUND Y)
         o := ((b1*pi)*((j1+a1)/Y1))
        z := (r1*COS(o))+c1
         rays1 := (REAL32 ROUND rays)
        z1 := z/(b1/rays1)
        ZZ := (INT TRUNC z1)

       IF
        j>0
          E[j-1][i] := ZZ
```

```
      TRUE
        E[Y-1][i] := ZZ
      G[j][i] := ZZ
}}}

SEQ jj=0 FOR views
 SEQ ii=0 FOR rays
 SEQ
    input ? con[jj][ii]
   output ! Q[0][0]
   {{{ shiftQ
  SEQ
     SEQ j=1 FOR (Y-1)
      SEQ i=0 FOR B
        H[j][i] := Q[j-1][i]
      H[0][0]:=con[jj][ii]

     SEQ j=0 FOR Y
      SEQ i=0 FOR B
        Q[j][i]:=H[j][i]
   }}}
   {{{ checkG
  SEQ
    SEQ j=0 FOR Y
     SEQ i=0 FOR B
     SEQ
      IF
         G[j][i]=m[j][i]
           F[j][i]:=F[j][i]+Q[j][i]
        TRUE
         SKIP
   }}}
   {{{ incM
  SEQ
    SEQ j=0 FOR Y
     SEQ i=0 FOR B
     SEQ
        m[j][i]:=m[j][i]+1
      IF
         m[j][i]>(rays-1)
        SEQ
          m[j][i]:=0
           E[j][i]:=G[j][i]
         IF
          j=0
             G[j][i]:=E[Y-1][i]
          TRUE
             G[j][i]:=E[j-1][i]
        TRUE
         SKIP
   }}}

SEQ ii=0 FOR rays
 SEQ
  output ! Q[0][0]
  {{{ shiftQ
 SEQ
   SEQ j=1 FOR (Y-1)
```

```
        SEQ i=0 FOR B
          H[j][i] := Q[j-1][i]
       H[0][0]:=0

       SEQ j=0 FOR Y
        SEQ i=0 FOR B
          Q[j][i]:=H[j][i]
     }}}
     {{{ checkG
     SEQ
      SEQ j=0 FOR Y
       SEQ i=0 FOR B
       SEQ
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
         TRUE
          SKIP
     }}}
     {{{ incM
     SEQ
      SEQ j=0 FOR Y
       SEQ i=0 FOR B
       SEQ
          m[j][i]:=m[j][i]+1
        IF
           m[j][i]>(rays-1)
         SEQ
           m[j][i]:=0
            E[j][i]:=G[j][i]
          IF
           j=0
              G[j][i]:=E[Y-1][i]
           TRUE
              G[j][i]:=E[j-1][i]
        TRUE
         SKIP
     }}}

   SEQ j=0 FOR Y
    SEQ
     SEQ i=0 FOR B
      output ! F[j][i]
     SEQ i=0 FOR X
     SEQ
       send ? F[j][i]
       receive ! F[j][i]
:
}}}F

}}}
{{{ SC backprojection
:::A 3 10
{{{F backprojection
:::F backpr01.tsr
#USE strings
#USE ufiler
#USE filerhdr
```

```
#USE uservals
#USE interf
#USE userio
#USE snglmath


PROC backprojection(CHAN OF ANY in, out)
 [120][65]INT F,m,Q,E,G,H:
 INT a,b,ZZ,zz,any,w,u,uu,i0,i2:
 REAL32 r1,o,z,z1,i1,X1,j1,Y1,rays1:
 {{{ vals
 VAL X IS 65:
 VAL A IS 4:
 VAL Y IS 120: --number of sectors
 VAL rays IS 65:
 VAL views IS 60:
 VAL M IS 3900:
 VAL pi IS 3.1415926(REAL32):
 VAL a1 IS 0.5(REAL32):
 VAL b1 IS 2.0(REAL32):
 VAL c1 IS 1.0(REAL32):
 }}}

 SEQ
  in ? i2
  out ! (i2+A)
  {{{ initialize
 SEQ
   SEQ j=0 FOR Y
    SEQ i=i2 FOR A
    SEQ
      F[j][i] := 0
       a := (5*rays)-(j+i)
      b := a/rays
       m[j][i] := a-(b*rays)
      Q[j][i] := 0
       i1 := (REAL32 ROUND i)
       X1 := (REAL32 ROUND X)
       r1 := (i1+a1)/X1
       j1 := (REAL32 ROUND j)
       Y1 := (REAL32 ROUND Y)
       o := ((b1*pi)*(j1+a1)/Y1))
       z := (r1*COS(o))+c1
       rays1 := (REAL32 ROUND rays)
       z1 := z/(b1/rays1)
       ZZ := (INT TRUNC z1)

     IF
      j>0
        E[j-1][i] := ZZ
      TRUE
        E[Y-1][i] := ZZ
     G[j][i] := ZZ
 }}}

 SEQ jj=0 FOR views
  SEQ ii=0 FOR rays
  SEQ
```

```
    in ? u
     out ! Q[0][((i2+A)-1)]
     {{{ shiftQ
    SEQ
      SEQ j=1 FOR (Y-1)
       SEQ i=i2 FOR A
         H[j][i] := Q[j-1][i]
      SEQ i=(i2+1) FOR (A-1)
       H[0][i]:=Q[0][i-1]
      H[0][i2]:=u

      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
         Q[j][i]:=H[j][i]
    }}}
     {{{ checkG
    SEQ
      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
       SEQ
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
         TRUE
          SKIP
    }}}
     {{{ incM
    SEQ
      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
       SEQ
          m[j][i]:=m[j][i]+1
        IF
           m[j][i]>(rays-1)
          SEQ
            m[j][i]:=0
             E[j][i]:=G[j][i]
           IF
            j=0
               G[j][i]:=E[Y-1][i]
             TRUE
               G[j][i]:=E[j-1][i]
         TRUE
          SKIP
    }}}
  SEQ ii=0 FOR rays
   SEQ
    in ? uu
     out ! Q[0][((i2+A)-1)]
     {{{ shiftQ
    SEQ
      SEQ j=1 FOR (Y-1)
       SEQ i=i2 FOR A
         H[j][i] := Q[j-1][i]
      SEQ i=(i2+1) FOR (A-1)
       H[0][i]:=Q[0][i-1]
      H[0][i2]:=uu
```

```
      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
        Q[j][i]:=H[j][i]
    }}}
    {{{ checkG
    SEQ
      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
       SEQ
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
          TRUE
           SKIP
    }}}
    {{{ incM
    SEQ
      SEQ j=0 FOR Y
       SEQ i=i2 FOR A
       SEQ
          m[j][i]:=m[j][i]+1
        IF
           m[j][i]>(rays-1)
          SEQ
            m[j][i]:=0
            E[j][i]:=G[j][i]
           IF
            j=0
               G[j][i]:=E[Y-1][i]
            TRUE
               G[j][i]:=E[j-1][i]
          TRUE
           SKIP
    }}}

  SEQ j=0 FOR Y
   SEQ
    SEQ i=0 FOR i2
     in ? F[j][i]
    SEQ i=0 FOR (i2+A)
     out ! F[j][i]
:
}}}F

}}}
{{{ SC back16
:::A  3 10
{{{F back16
:::F back1600.tsr
#USE strings
#USE ufiler
#USE filerhdr
#USE uservals
#USE interf
#USE userio
#USE snglmath
```

```
PROC back16(CHAN OF ANY in, out)
 [120][65]INT F,m,Q,E,G,H:
 INT a,b,ZZ,zz,any,w,u,uu,i0,i2:
 REAL32 r1,o,z,z1,i1,X1,j1,Y1,rays1:
 {{{  vals
 VAL X IS 65:
 VAL A IS 4:
 VAL Y IS 120: --number of sectors
 VAL rays IS 65:
 VAL views IS 60:
 VAL M IS 3900:
 VAL pi IS 3.1415926(REAL32):
 VAL a1 IS 0.5(REAL32):
 VAL b1 IS 2.0(REAL32):
 VAL c1 IS 1.0(REAL32):
 }}}

 SEQ
  in ? i2
  {{{  initialize
 SEQ
   SEQ j=0 FOR Y
    SEQ i=i2 FOR A
    SEQ
      F[j][i] := 0
       a := (5*rays)-(j+i)
      b := a/rays
       m[j][i] := a-(b*rays)
      Q[j][i] := 0
       i1 := (REAL32 ROUND i)
       X1 := (REAL32 ROUND X)
       r1 := (i1+a1)/X1
       j1 := (REAL32 ROUND j)
       Y1 := (REAL32 ROUND Y)
       o := ((b1*pi)*((j1+a1)/Y1))
       z := (r1*COS(o))+c1
       rays1 := (REAL32 ROUND rays)
       z1 := z/(b1/rays1)
       ZZ := (INT TRUNC z1)

     IF
      j>0
        E[j-1][i] := ZZ
      TRUE
        E[Y-1][i] := ZZ
      G[j][i] := ZZ
 }}}

 SEQ jj=0 FOR views
  SEQ ii=0 FOR rays
  SEQ
    in ? u
    {{{  shiftQ
   SEQ
     SEQ j=1 FOR (Y-1)
      SEQ i=i2 FOR A
        H[j][i] := Q[j-1][i]
      SEQ i=(i2+1) FOR (A-1)
```

```
      H[0][i]:=Q[0][i-1]
     H[0][i2]:=u

     SEQ j=0 FOR Y
      SEQ i=i2 FOR A
       Q[j][i]:=H[j][i]
   }}}
   {{{ checkG
   SEQ
     SEQ j=0 FOR Y
      SEQ i=i2 FOR A
      SEQ
       IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
         TRUE
          SKIP
   }}}
   {{{ incM
   SEQ
     SEQ j=0 FOR Y
      SEQ i=i2 FOR A
      SEQ
         m[j][i]:=m[j][i]+1
       IF
          m[j][i]>(rays-1)
         SEQ
           m[j][i]:=0
            E[j][i]:=G[j][i]
          IF
           j=0
              G[j][i]:=E[Y-1][i]
           TRUE
              G[j][i]:=E[j-1][i]
        TRUE
         SKIP
   }}}
SEQ ii=0 FOR rays
 SEQ
  in ? uu
  {{{ shiftQ
  SEQ
    SEQ j=1 FOR (Y-1)
     SEQ i=i2 FOR A
      H[j][i] := Q[j-1][i]
    SEQ i=(i2+1) FOR (A-1)
     H[0][i]:=Q[0][i-1]
    H[0][i2]:=uu

    SEQ j=0 FOR Y
     SEQ i=i2 FOR A
      Q[j][i]:=H[j][i]
  }}}
  {{{ checkG
  SEQ
    SEQ j=0 FOR Y
     SEQ i=i2 FOR A
     SEQ
```

```
        IF
          G[j][i]=m[j][i]
            F[j][i]:=F[j][i]+Q[j][i]
          TRUE
          SKIP
      }}}
      {{{ incM
      SEQ
        SEQ j=0 FOR Y
        SEQ i=i2 FOR A
        SEQ
          m[j][i]:=m[j][i]+1
        IF
          m[j][i]>(rays-1)
          SEQ
            m[j][i]:=0
            E[j][i]:=G[j][i]
          IF
            j=0
              G[j][i]:=E[Y-1][i]
            TRUE
              G[j][i]:=E[j-1][i]
        TRUE
        SKIP
      }}}

    SEQ j=0 FOR Y
     SEQ
      SEQ i=0 FOR i2
       in ? F[j][i]
      SEQ i=0 FOR (i2+A)
       out ! F[j][i]
:
}}}F

}}}
{{{  configuration
{{{  link constants
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:
}}}
{{{  vals
VAL P IS 17:
}}}
{{{  CHANs
[P]CHAN OF ANY pipe:
CHAN OF ANY app.in, app.out:
}}}

PLACED PAR
 PROCESSOR 0 T4
   PLACE app.in AT link0in:
```

```
    PLACE app.out AT link0out:
    PLACE pipe[0] AT link1out:
     PLACE pipe[P-1] AT link3in:
    interface ( app.in, pipe[0], pipe[P-1], app.out)

  PLACED PAR s=1 FOR (P-2)
   PROCESSOR s T4
     PLACE pipe[s-1] AT link0in:
     PLACE pipe[s] AT link1out:
     backprojection ( pipe[s-1], pipe[s] )

  PROCESSOR (P-1) T4
    PLACE pipe[P-2] AT link0in:
    PLACE pipe[P-1] AT link2out:
    back16 (pipe[P-2], pipe[P-1])
}}}
```