Parallel Implementation of the Filtered Back Projection Algorithm for Tomographic Imaging

by

Raman P.V. Rao

Project Report submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Dr. A. Lynn Abbott, Co-chairman

Dr. Ronald D. Kriz, Co-chairman

Dr. Calvin J. Ribbens

January, 1995

Blacksburg, Virginia

Parallel Implementation of the Filtered Back Projection Algorithm for Tomographic Imaging

by

Raman P.V. Rao

Committee Co-chairmen:

Dr. A. Lynn Abbott

The Bradley Department of Electrical Engineering

and

Dr. Ronald D. Kriz

Department of Engineering Science and Mechanics

(ABSTRACT)

Computer Tomography(CT) is used in several applications — medicine, non-destructive testing/evaluation, astronomy and others to look inside objects and analyze internal structures. However, the problem in general is computationally very intensive. The large computational requirement has led to large times for CT reconstruction. This has hindered the use of CT in many applications where CT image data is required in real time. Parallel processing is one of the techniques available which can reduce the reconstruction time very significantly. In this project, we describe a parallel implementation of the filtered back projection algorithm for finding 2D cross sectional images. The results show that parallel processing is very viable for CT reconstruction and results in significant run time savings. An implementation on two different machines is described - the Intel Paragon and the Connection Machine. For comparison, the filtered back projection algorithm was first programmed as a serial algorithm using Matlab software on a Sun Sparc 10. It has been found that while the serial program in Matlab takes approximately 12 minutes for a reconstruction, the same data can be parallelized on the Intel Paragon in about 2.2 secs and on the CM5 in about 3 secs.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ronald D. Kriz for his guidance, support and encouragement throughout this project. My discussions with him were crucial in the formulation and later, the implementation of this project. I also thank Dr. Lynn Abbott for agreeing to be my Co-chairman, and for his guidance and support which shaped this report. I would like to thank Dr. Calvin Ribbens for agreeing to be on my committee, and for useful discussions on parallel computation in and outside the classroom.

I would like to acknowledge my thanks to Dr. Chris Henze, Department of Biology for spending many hours discussing with me the CT geometry, algorithms and his own implementations. My discussions with him were crucial for the success of this project.

I thank Dr. Lane Watson and the NCSA (University of Illinois) for permission to use the Intel Paragon and Connection Machine supercomputers respectively for this research.

I thank NASA Langley Research Center, and especially Dr. William Winfree for sponsoring this research under the NASA grant number NAS1-1847 task #49.

I would like to thank Dr. Kenneth Reifsnider for the use of his facilities during my stay at MRG (Materials Response Group). I also thank all members of MRG (faculty, staff and students) for many a pleasant time that I spent here.

Special thanks to secretaries Melba, Shelia and Loretta Estes (Electrical Engineering) for their help in administrative matters.

On a personal note, I would like to thank my wife Sailaja for her love and patience all these years. Riding through tough times has certainly been easier because of her.

Finally I thank my parents for their love and encouragement all through my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 Introduction

Computer Tomography(CT) refers to the cross-sectional imaging of an object from its projection data. The mathematical basis for CT was first discovered by Radon in 1917 [1]. However, it was not until 1972 that the first CT scanner was invented, for which G.N. Hounsfield and Alan McCormack received the Nobel Prize [2]. Since then, a many advances have been made in scanner technology as well as in the algorithms used for CT reconstruction. Today CT scanners are used extensively in many applications (medicine, astronomy and non-destructive testing) to carry out three dimensional volume visualization of objects [3, 4].

Fundamentally, tomography deals with reconstruction of an object from its projections. The technique of tomography consists of passing a series of rays (in parallel, fan or cone formation) through an object, and measuring the attenuation in these rays by placing a series of detectors on the receiving side of the object. These measurements are called projections. This data is collected at various angles from 0 to 360 degrees. Any of the several algorithms available can then be used to reconstruct its 2D cross-sectional image from its projections. The property that is actually computed is the linear attenuation coefficient of that object, at various points in the object's cross-section. If this process is repeated at various heights along the object, we obtain several 2D cross-sectional images, which can then be stacked one on the top of another to get internal 3D volume vizualization of the object. There are many methods of collecting the projection data. The first is called "parallel projection"

and is illustrated in fig. (1.1).

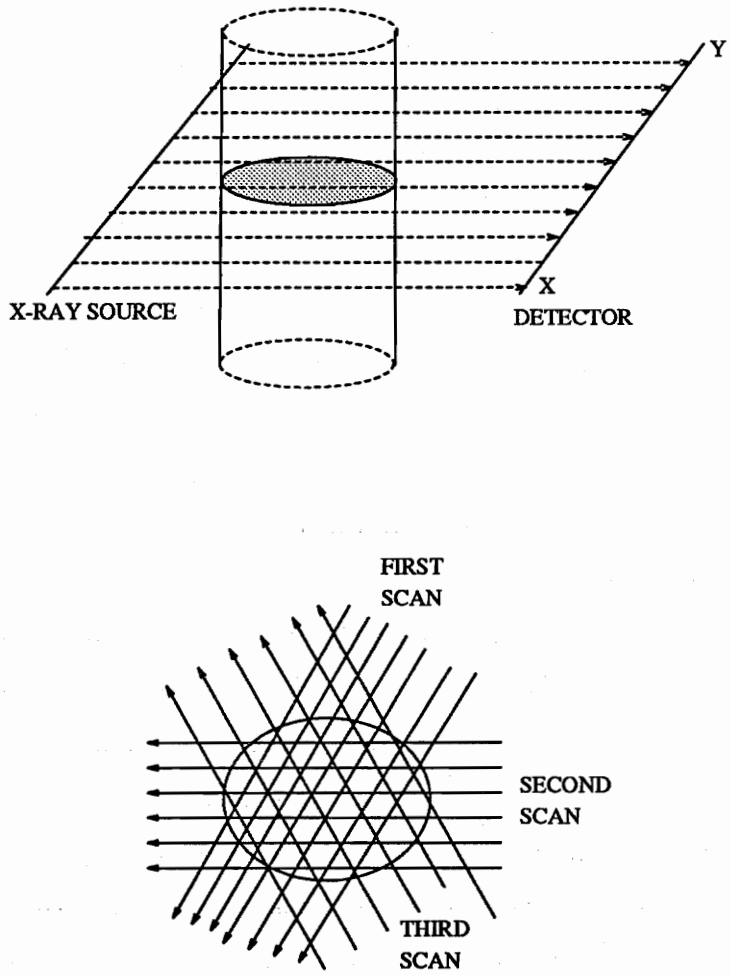

Figure 1.1: A simple scanning system.

In fig. (1.1) an x-ray source transmits rays which pass through the object and are received by the detector. The source and detector are moved linearly along X-Y (fig (1.1)) and projection data can be collected along X-Y. The object can then be rotated by some fixed angle, and this process can be repeated. Reconstructing this projection data over all the angles would then lead to a cross-sectional image. The same process can be repeated with multiple sources and detectors and is much faster as the source-detector geometry

does not have to move to collect the projection data in the same angle. Here, either the source detector geometry or the object can be rotated at a fixed angles (intervals) and the projection data can be collected again. This arrangement leads to faster collection of projection data. The method of fig. (1.1) is called parallel projection because the rays are parallel to each other when passing through the object. There are other methods for collecting the projection data such as fan beam and cone beam projections as shown in figs. (1.2) and (1.3). In fan beam, the point source of radiation is used to collect data along one slice at each angle. The point source (or the object) can then be rotated to get other projections from 0 to 360 degrees. Cone beam projections can be used to generate projection data for 3D volume vizualization of objects within a very short time. However, cone beam projections are not used often due to errors in measurements which come up due to the cone nature of the rays. Even fan-beam measurements have some error, but this is usually within tolerable limits. The safest method of generating the projection data is by using parallel rays which has been shown to have the least error [3, 5, 4]. Also, CT algorithms for fan and cone beam projections are more complicated computationally. In this project, we have considered the parallelization for parallel beam projection data only. The same algorithm can be used for fan beam projections as well with some changes.

Serial implementation of the filtered back projection algorithm is time consuming and does not give the images in real time. We have parallelized the serial algorithm to achieve a very significant speed up as compared to the serial algorithm (minutes vrs. a few seconds). The method adopted in the parallelization has been row-wise distribution of the data - and then carrying out the same instructions on the different data. The row-wise distribution results basically in the distribution of the projection data for different angles on different processors.

This project describes the implementation of the filtered back-projection algorithm on a multiprocessor Intel Paragon and the Connection Machine (CM5). Implementation of the

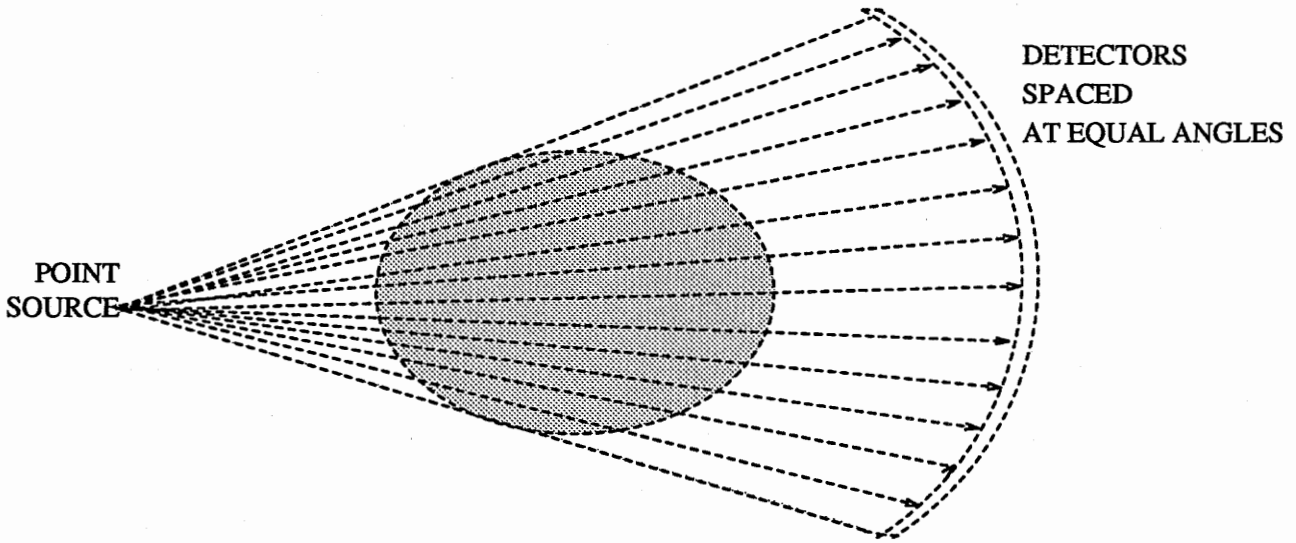

Figure 1.2: Fan beam projection.

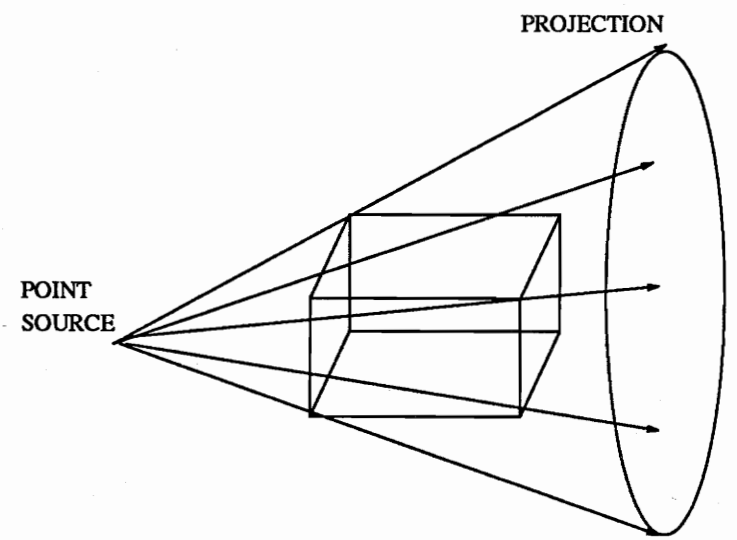

Figure 1.3: Fan beam projection.

algorithm on these machines has led to new algorithms for parallel implementation, and these are described in this report. It is seen that parallel processing leads to significant improvements in the computing time requirements.

1.2 Literature Review

The mathematical basis for tomography was originally developed by Radon in 1917 [1]. Then, there was dormancy in the field until the invention of the first CT scanner in 1972 by G.N. Hounsfield and Alan McCormack [2] which is considered to be a major breakthrough in the field of Computed Tomography. The filtered back-projection algorithm was invented by Bracewell and Riddle [6], and later independently with additional insights by Ramanchandran and Lakshminarayanan [3, 5], who are also responsible for the convolution back-projection algorithm. This is the algorithm now used by almost all commercially available CT scanners [3]. The filtered back-projection algorithm for fan beam data was developed by Lakshminarayanan [7]. Several researchers have proposed algebraic techniques for reconstruction: Gordon et.al. [8, 9], Herman [10, 11] and Budinger [12] . Wernecke and D'Addario [13] proposed a maximum entropy method for reconstruction. It was shown by Shepp and Logan [14] that the filtered back-projection method was much superior to other methods (especially the algebraic methods) in 1974.

Several methods for improving the speed of the algorithms have been proposed. These can broadly be classified in three categories: (a) Improvements in the algorithms, (b) Improvements in the hardware used for computing the reconstructions, and (c) Parallel processing techniques. Under category (a) we have the work of Tanaka [15], Kenue [16] , Wang [17] , Dreike [18] and several others. Peters [19] developed the STRETCH algorithm which used precomputed look-up tables for interpolation.

Improvements in hardware, and the use of dedicated hardware was suggested by several

workers: Thompson et. al. [20] implemented special hardware to implement the STRETCH algorithm. Hartz et. al. [21] have implemented a special hardware for carrying out back-projection on an event by event basis with $2\mu s$ per event in positron CT. More recently, Agi et. al. [22] have come up with the VLSI implementation of a dedicated microprocessor for carrying out forward and backward Radon transform. They make extensive use of pipelining to achieve faster processing. It has been shown that 64 of these Application Specific Integrated Circuits (ASIC's) will backproject 512x512 pixel raster-scan image from filtered CT data in 70 ms [22]. Although ASIC's have advantages in many applications, in this case since multiprocessing is required, it becomes less practical to use an ASIC as they offer less flexibility to the user.

Parallel implementation of the CT algorithms became important when it became necessary to reconstruct 3D images in real time[23]. Studies have indicated that parallel implementation is the only way to implement 3D reconstruction algorithms in real time [23, 24, 25]. Jones et. al. [26] proposed a scheme for parallelizing the Expectation Maximization algorithm. Pixels are circulated through the processing elements(PE's) , each of which backprojects one view of the projection data. Lacer et. al. [27] have proposed a multiprocessor scheme for the Maximum Liklehood Estimator(MLE) based algorithm. All the matrices are partitioned into disjoint segments which are stored in different sections of memory. Each PE has direct access to one of these memory sections - and the other PE's can access the same through a cross-bar switch. The use of a cross-bar switch limits the maximum number of processors which can be used, as the cost of switching increases very fast for large number of processors. Both the above parallel implementations discussed are for iterative type of algorithms. Chen et. al. [23] have implemented the filtered back-projection algorithm on an Intel iPSC/2 - a hypercube system with 32 nodes. Convolution was done in the spatial domain (instead of the frequency domain) to reduce the memory requirements (as zero padding of data has to be done to reduce errors). The projection data as well as the response function were distributed to all the nodes; convolution was

done in parallel by carrying out all the multiplications on different processors in parallel. Then the intermediate results are sent to a designated node(PE) where the additions were performed. In the back-projection stage, they used the 3D incremental algorithm developed by Cho et. al. [24]. Here back-projection is performed on a beam by beam basis by each PE, and the final image is collected by a designated processor. Broadcasting and integration (of messages) can be done in O(logN) steps. But a major disadvantage of their method seems to be that a lot time is spent in interprocessor communication, as even the convolution is done in parallel. Also, it seems that carrying out convolution in the spatial domain (and not in the frequency domain where fast algorithms like the FFT can be used) is also a waste of computing power. Atkins et. al. [28] have implemented the STRETCH algorithm for use in 3D PET (positron emission tomography). The transputers (T800's) were connected in a tree type of arrangment with one node working as a master and the others as workers. They used the same transputers for speeding up the data acquisition, image reconstruction and display [28, 29]. They have estimated that 3D images could be reconstructed in about 10 minutes using about 200 nodes (T800's) in a tree type of interconnection. Barresi et al. have also implemented the 3D filtered back-projection algorithm on Transputers(T800) using the OCCAM programming language [25]. They tested their algorithms on a 5 transputer network, and they also used parallel FFT algorithms for finding the FFT. They have reported a reconstruction frequency of 700Hz with 5 T800 Transputers.

In this project, we have implemented the filtered back-projection algorithm for 2D on two different parallel platforms: a 28 node Intel Paragon with the nodes connected as a mesh, and a CM5 (Connection Machine). The Paragon implementation is fully complete with message passing controlled by the parallel algorithm. Although both the implementations have been done in high level languages (using the parallel constructs provided by the language compilers on those machine), the CM5 software is parallelized by the Connection Machine compilers, and so we do not have explicit control on the message passing (unlike the Intel Paragon). On the Intel Paragon we have implemented an explicit message passing

algorithm using the message passing constructs provided on the machine. The parallel times were compared with the serial algorithm in Matlab.

Chapter 2

The Filtered Back Projection Algorithm

2.1 Projections

The Filtered Back Projection algorithm [5] uses Fourier theory to arrive at a closed form solution to the problem of finding the linear attenuation coefficient at various points in the cross-section of an object. A fundamental result linking Fourier transforms to cross-sectional images of an object is the Fourier Slice Theorem [3, 4]. In this report we will be concerned only with parallel beam projection data. The Fourier Slice Theorem for the parallel beam projection data is given here. The same justifications can be made for fan beam and cone beam projection data [3, 4, 5, 7].

Let x, y represent the coordinates inside the object(fig. (2.1)), and $f(x, y)$ the density (attenuation coefficient) in rectangular co-ordinates of the object under consideration at the cross-section at which the imaging has to be done. Let $P_\theta(t)$ represent the projection of the object at distance t from the center. The equation of the line AB is $x\cos(\theta) + y\sin(\theta)$. Then, the projections $P_\theta(t)$ are defined as [3]:

$$P_\theta(t) = \int_{(\theta,t)line} f(x, y)ds \tag{2.1}$$

It has been shown [3] that the above equation can be written using a delta function as:

$$P_\theta(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\,[\delta(x\cos(\theta) + y\sin(\theta) - t)\;dx\;dy] \tag{2.2}$$

The function $P_\theta(t)$ is known as the Radon transform of $f(x, y)$ [3]. A set of $P_\theta(t)$'s for constant θ are the projections of the object at the cross-section where the rays are being

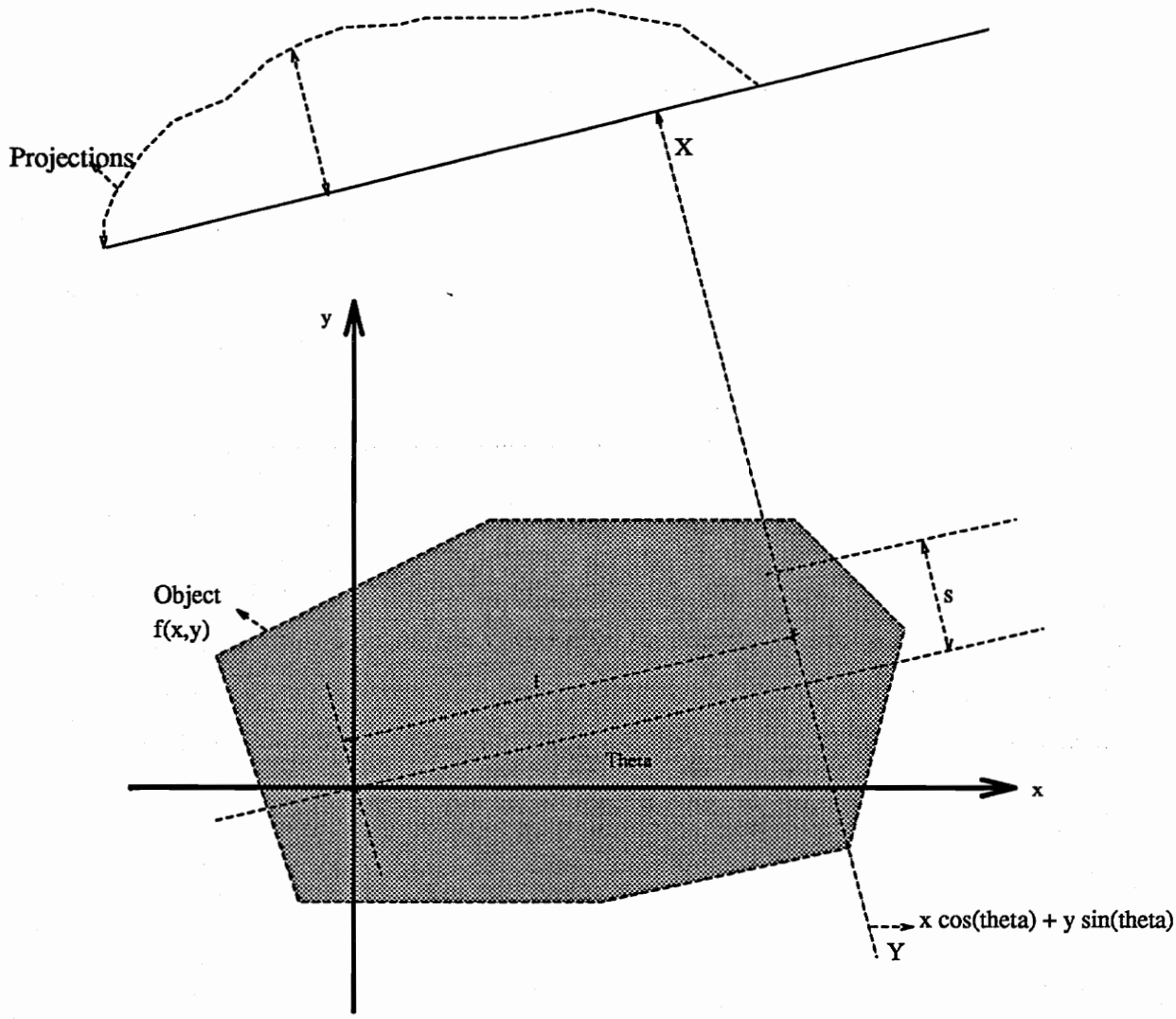

Figure 2.1: Formation of Projections

passed. When θ is held constant throughout a projection, we get parallel projection data. If a point source is used, we get fan beam projection data. The name *fan beam* results from the fact that the line integrals are measured along fans [3, 4].

2.2 Fourier Slice Theorem

An important result linking Fourier theory to the projections was developed by Bracewell [6], Ramachandran and Lakshminarayanan [5, 7]. The following proof is a result of Kak and Slaney [3]. The 2D Fourier transform is defined as:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \tag{2.3}$$

where u, v are measured in cycles/unit length. From the definition of the 1D fourier transform, we can find the Fourier Transform of the projection data $P_\theta(t)$ at an angle θ as:

$$S_\theta(w) = \int_{-\infty}^{\infty} P_\theta(t)e^{-j2\pi wt}dt \tag{2.4}$$

where, w is the in radians/unit length.

We define a new co-ordinate system (t, s) defined by the rotation of the (x, y) system by the angle θ such that

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.5}$$

In the (t, s) co-ordinate system, a projection would then be defined as

$$P_\theta(t) = \int_{-\infty}^{\infty} f(t,s)\ ds \tag{2.6}$$

From eqns. (2.4) and (2.5), we can say that

$$S_\theta(w) = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(t,s)ds\ e^{-j2\pi wt}\right]dt \tag{2.7}$$

This can be transformed into the (x, y) co-ordinate system using the result of eqn. (2.5) to get the following result:

$$S_\theta(w) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi wt}dxdy \tag{2.8}$$

where,

$$t = x\cos(\theta) + y\sin(\theta) \tag{2.9}$$

The right hand side of eqn. (2.8) represents the two-dimensional Fourier transform of the density f(x,y) and left hand side is the 1D Fourier transform of the projections $P_\theta(t)$. Therefore, taking the 1D Fourier transform of the projections of an object at an angle θ is equivalent to obtaining the two dimensional Fourier transform of the density f(x,y) along the line t inclined at an angle θ. Therefore if we take these projections at many angles θ_1,θ_2,θ_3 ,θ_4 etc., then we can get this 2D Fourier transform of the projections at many such lines t_1, t_2, t_3,... inclined at various angles. If the number of angles is large enough, we will get many lines of 2D Fourier transforms of the object. If we now find the inverse fourier transform of all these lines, we get the object's densities f(x,y) for all (x,y) in the object's cross-section. That is,

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S_\theta(w)e^{j2\pi(ux+vy)}dudv \tag{2.10}$$

represents the back projection along the line t,

where,

$$S_\theta(w) = F(w\cos\theta, w\sin\theta) = F(u,v) \tag{2.11}$$

This is the Fourier Slice Theorem, and in essence provides a justification for using the Fourier theory in CT algorithms. A more detailed proof can be found in [3], [5] and [6].

2.3 The Filtered Back Projection Algorithm

Proof of the filtered back projection algorithm follows from eqn. (2.10) [3, 5, 7, 4]. The following proof is due to Kak and Slaney [3]. If the co-ordinate system in the frequency domain (u,v) which is rectangular is changed to the *polar* co-ordinate system, we have to

make the following substitutions:

$$u = w\cos(\theta),\ v = w\sin(\theta) \tag{2.12}$$

where, w = radius and θ = angle in radians and the differentials change as

$$dudv = wdwd\theta \tag{2.13}$$

$$f(x,y) = \int_0^{2\pi}\int_0^{\infty} F(w,\theta)e^{j2\pi w(x\cos\theta + y\sin\theta)}wdwd\theta \tag{2.14}$$

We can split the above integral by considering θ from 0 radians to π radians, and then from π radians to 2π radians. We then get,

$$\begin{aligned} f(x,y) = &\int_0^{\pi}\int_0^{\infty} F(w,\theta)e^{j2\pi w(x\cos\theta + y\sin\theta)}wdwd\theta + \\ &\int_{\pi}^{2\pi}\int_0^{\infty} F(w,\theta+\pi)e^{j2\pi w(x\cos(\theta+\pi)+y\sin(\theta+\pi))}wdwd\theta \end{aligned} \tag{2.15}$$

But it is known from Fourier theory that [3, 5, 7]

$$F(w,\theta+\pi) = F(-w,\theta) \tag{2.16}$$

Therefore simplifying eqn. (2.15) we get:

$$f(x,y) = \int_0^{\pi}\int_{-\infty}^{\infty}\left[F(w,\theta)\ |w|\ e^{j2\pi(x\cos\theta + y\sin\theta)}dw\right]d\theta \tag{2.17}$$

and $x\cos\theta + y\sin\theta = t$ as defined earlier.

From (2.4) and (2.7), we can say that in this case, $F(w,\theta)$ is the same as $S_\theta(w)$ of Eqn. (2.7). Therefore,

$$f(x,y) = \int_0^{\pi}\left[\int_{-\infty}^{\infty} S_\theta(w)\ |w|\ e^{j2\pi(x\cos\theta + y\sin\theta)}dw\right]d\theta \tag{2.18}$$

In the Eqn. (2.18) above, the terms inside the square brackets (the operation indicated by the inner integral) represents a filtering operation and evaluate the *filtered projections*, and the operation being performed by the outer integral evaluate the *back-projections*, which

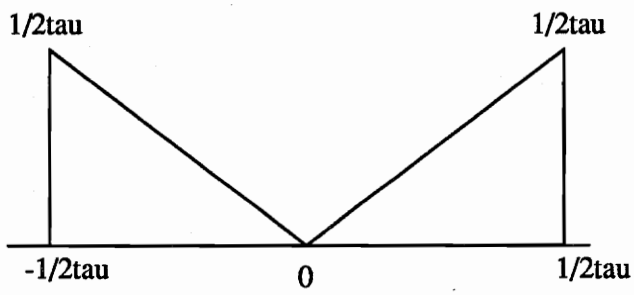

Figure 2.2: The response function in frequency domain.

basically represents a smearing of the filtered projections back on to the object and then finding the mean over all the angles.

The filtered back projection algorithm can therefore be thought of as a three step process:

1. Finding the Fourier Transform(FT) in 1D of the projections.

2. Finding the *filtered projections.* This essentially means multiplying the results of step 1. with a response function whose function looks as shown in fig. (2.2) in the frequency domain, and then finding the inverse Fourier Transform (IFFT) . This step is essentially the same as carrying out convolution in the time domain. It can be represented mathematically as

$$Q_\theta(t) = \int_{-\infty}^{\infty} S_\theta(w)\ |w|\ e^{i2\pi wt} dw \tag{2.19}$$

3. Finding the *back* projections. This step is the smearing of the filtered projections back on to the object, and is mathematically represented by

$$f(x,y) = \int_0^\pi Q_\theta(x\cos\theta + y\sin\theta) d\theta \tag{2.20}$$

These three steps represent the filtered back projection algorithm. In the discrete domain, the algorithm changes only slightly. The important steps are outlined below:

1. Find the 1D FT of the projections for each angle.

2. Multiply the result of step 1. above with the response function in the frequency domain (equivalent to convolving with the response function in the time domain). In the actual simulations, the response function is simply a ramp with 45° slope and with the same length as the final reconstructed image. (In the function shown in fig. (2.2), *tau* represents the distance between the rays).

3. Find the IFFT of the results in step 2. This gives us the *filtered* projections in the discrete domain and correspond to $Q_{\theta_i}(n)$, where θ_i are the various angles at which the projections were taken, and n is the ray number at which the line projection was taken.

4. Back-project. The integral of the continuous time system now becomes a summation, and we get

$$f(x,y) = \sum_{i=1}^{K} Q_{\theta_i}(x\cos(\theta_i) + y\sin(\theta_i)) \tag{2.21}$$

It should be noted here that (x,y) are chosen by the program while back projecting. So the value of $x\cos(\theta) + y\sin(\theta)$ may not correspond exactly to a value of n for the filtered projections which may have been calculated in the previous step. Therefore interpolation has to be done, and usually linear interpolation is quite sufficient [3, 4]. It may be noted that in terms of computational time, this step consumes the maximum time (about 80%). Many authors have proposed different strategies for coping with this computational demand. One popular method is to find the filtered projections for a very large number of points by heavily zero-padding the original data. Then, while back-projecting, the nearest or closest point can be selected. In our implementation, we carry out the linear interpolation to get realistic estimates of the speed of the linear and parallel algorithms. We however, zero-pad the data to four times its length to remove the *dishing* and *dc artifacts* from the final images. These artifacts are a result of the finite number of frequencies which can be represented in the discrete Fourier transform(DFT) [3], and it is usually recommended to zero-pad to double

the length of the vectors to reduce dishing and dc artifacts [3, 30]. Zero-padding gives a better resolution in the frequency domain and the Fast Fourier Transform (FFT) algorithm normally used in computing the DFT is more accurate if larger point sizes are used [3, 30].

We now clarify some CT terminology. A *bin* is a detector where a ray was received. The *mid bin* refers to the point where the ray should have been received if the ray corresponding to a particular x and y points had a detector. However, detectors (bins) are just spaced at equal intervals, and so all the points inside the object do not have a detector in the exact location of the *mid bin.* The nearest bin lower than the mid bin is called the *low bin* and the nearest bin higher than the mid bin is called the *high bin.* This is the terminology used in CT parlance. Generally, from the mid bin, the values of the energy going into the low and high bins is evaluated by linear interpolation [3, 4].

In order to match the object and the image space, we normalize the (x,y) space into which the object is being back-projected so as to lie in the interval (-1,1). So the the entire back-projection space is normalized to lie in the imaginary square in the space (-1,-1) to (1,1) If this is done, the factor t now becomes

$$t = ((x - x_{cen})/x_{cen})\cos(\theta_i) + ((y - y_{cen})/y_{cen})\sin(\theta_i) \tag{2.22}$$

Then the mid-bin is calculated by

$$mb = t * mp + mp \tag{2.23}$$

where mp is the midpoint of the number of rays. The mid-bin essentially represents the point where the ray in question actually was received. If there was no detector at that point, we assume for simplicity that, some parts of the energy would have been received by the two detectors (bins) nearest to the mid bin. We assume that this is proportional to the distance from the bin; i.e., if the mid-bin happened to be closer to the low-bin, then more of the energy would have been received by the low bin and less by the high bin.

This kind of assumption is essential for reconstructing the image - and is in essence the linear interpolation we talked about earlier. The low-bin and the high-bins can then be calculated as usual, i.e., $lb = truncate(mb)$ and $hb = lb + 1$. This method of finding the back-projections allows us to carry out the entire simulation without any knowledge of the distance between the rays, and the distance between the pixels (and what it represents in the actual image).

In our actual simulations, K=128, x_max_value=64 and y_max_value=64 were used. The original image which was forward projected was 16x16 and number of rays used to forward project were 16. With the above explanations, we can now define the *serial* algorithm to be as follows: We first define the terms which are used.

K: Number of angles θ_i at which the projections were taken.

NP: Number of Processors.

$P_{\theta_i}(n)$: The projections measured at an angle θ_i at the various points n (n varies from 1 to the total number of parallel rays used in grabbing the projections).

$H(n)$: This is called the "response function". It may be noted that $H(n)\ \forall\ n$ is a single row vector whereas $P_{\theta_i}(n)$ is a two dimensional matrix - the rows represent the various angles at which the measurements were taken and the columns in each row represent the projection data obtained for each ray at a certain angle. The response function that we use here is as shown in fig (2.2).

FFT Refers to the Fast Fourier Transform. FFT(X) refers to the Fast Fourier Transform of the vector X.

$IFFT$ Refers to the Inverse Fast Fourier Transform. IFFT(X) refers to the Inverse Fast Fourier Transform of the vector X.

$Q_{\theta_i}(n)$: Refered to as the 'filtered projections' of the measured projection values $P_{\theta_i}(n)$.

$f(x, y)$: refers to the linear attenuation coefficient at the point in the object indicated by the values of x and y.

With these conventions, we now define the Filtered Backprojection Algorithm to be as follows:

For i=1 to K

$Q_{\theta_i}(n)$ =IFFT((FFT of $P_{\theta_i}(n)$ with zero padding) x (FFT of $H(n)$ with zero padding))

End for

(All the FFT and IFFT operations are performed row wise with n going from 1 to total number of columns).

For x = 1,x_max_value

For y = 1,y_max_value

For i =1,K

1. Find $t = ((x - x_{cen})/x_{cen})\cos(\theta_i) + ((y - y_{cen})/y_{cen})\sin(\theta_i)$
2. mb = t*mp*tw +mp
3. lb = integer(mb); hb = lb+1
4. update the back-projection matrix for the calculated values of lb and hb.

End for

5. Evaluate, f(x,y) = $\frac{\pi}{K}\sum_{i=1}^{K} Q_{\theta_i}(xcos\theta_i + ysin\theta_i)$

   where K refers to the total number of angles over which the projections are taken , and, θ_i are the various angles at which projections measurements were taken.

End_for

End_for

Linear interpolation is carried out in step 4 in the above algorithm. Zero padding is done essentially to increase the accuracy of the FFT algorithm which has better accuracy if larger point sizes are used (zero padding a data sequence yields data points in the frequency domain which are more closely packed than the original [3, 30]).

Chapter 3

Parallel Implementation

3.1 Implementation on the Intel Paragon

3.1.1 Architecture of the Intel Paragon

The Intel Paragon is a mesh connected, distributed memory multicomputer. It can have more than 1000 nodes connected in the form of a mesh. The Intel Paragon supports general purpose MIMD (Multiple Instruction Multiple Data) as well as the SPMD (Single Program Multiple Data) type of programming. Each node which consists of an i860 processor, has its own memory and its own copy of the operating system running on it to support the message passing protocols which are required to run a program on multiple processors. Each processor operates on its own data. Therefore to share data between the nodes, it requires message(s) to be sent from one node to the other. In this project, we have adopted the SPMD style of programming for our implementation. The *NX message passing library* for this purpose has been provided by Intel with the machine. The Paragon also supports other message passing protocols like *PVM, P4* and *tcmsg.*

Of all the nodes, some of the nodes serve as service nodes and some as I/O nodes. This configuration is completely transparent to the user. The system at Virginia Tech (which was used for all the simulations in this report) has 28 nodes connected in the form of a mesh.

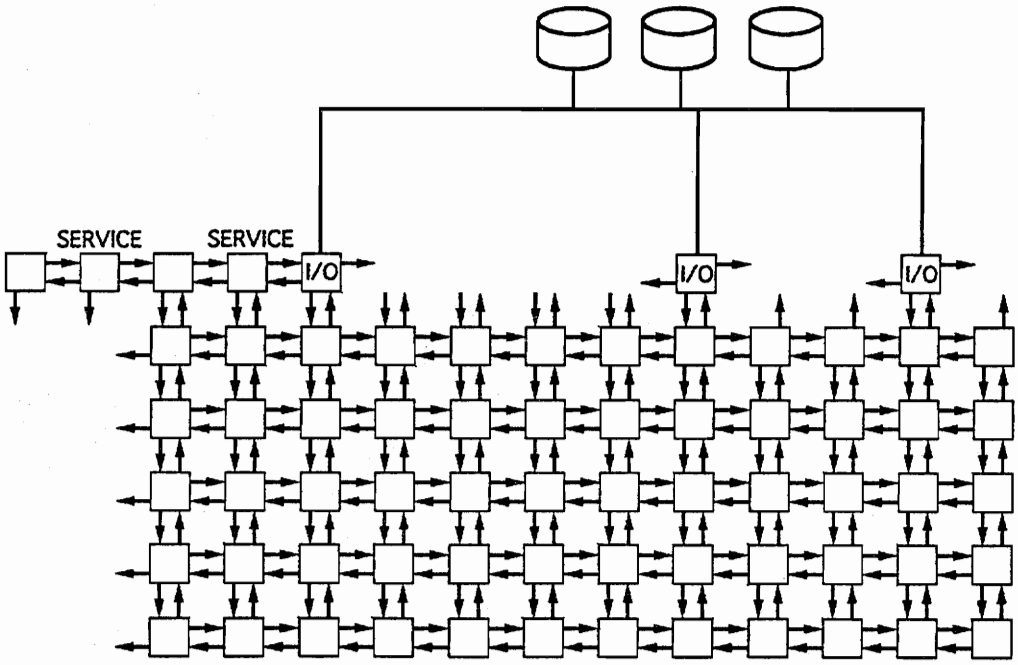

Figure 3.1: The Intel Paragon System overview.

3.1.2 Parallel Implementation

As mentioned earlier, the SPMD style of programming was adopted in our approach. This style is well suited to our problem for the following reasons:

1. The operations being performed on each row of the data are the same if we consider the original projections row-wise (for each angle). Hence, the SPMD syle is well suited in the calculation of the filtered projections.

2. In the back projection stage, we have found that it can be implemented with very little message passing between the processors. For example, we need to send only as many messages as the number of processors for carrying out the entire back projection. This is a low overhead on the system. Care has been taken to ensure that the number of times a communication is requested is as low as possible (even if the size of the data being transmitted is large) to keep the total communication setup time to a minimum.

3.1.3 Parallel Computation of Filtered Projections

Since the serial algorithm carries out the FFT computations row-wise on each of the matrices $P_{\theta_i}(n)$ $\forall$ $i = 1, .., K$ and $H(n)$, and, since the FFT operations on one row are independent from the other rows, it was decided to parallelize by distributing an equal number of rows to each processor. The projection data was generated by forward projecting a matrix of data which looked like a checker board. The size of the checker board was 16x16. The number of rows per processor was kept constant to support the SPMD style of programming. Since the data was an integer multiple of 2, speed up results are available for 1,2,4,8 and 16 processors. Since the FFT algorithm works best for vectors which are integer multiples of 2, zero padding has been done to keep the vector length of each row as an integer multiple of 2. Zero padding has the additional benefit of reducing the dishing and dc artifacts [3] inherent in the DFT algorithm. The FFT algorithm was run serially on all the processors, but each processor ran the algorithm on its own data. The maximum possible number of rays were used in all our simulations to project the data. For example, if the

original data was a checker board of size 16x16, 16 rays were used to project the data and if the original data was 32x32, 32 rays were used. Increasing the number of rays beyond the number of pixels does not add any additional information as the value of density between these pixels is unknown. It must be noted that in the case of real objects, increasing the number of rays would lead to some extra information. Kak has mentioned that even in the case of real objects, increasing the number of rays beyond a reasonable limit does not lead to images which are any better [3].

Hence parallel computation of the filtered projections is done as follows:

1. Find Number of rows per processor $= \frac{total\ number\ of\ rows}{number\ of\ processors}$
2. Initialize the H(n) and the projection matrix (PMAT) to zero to four times the number of rays. For example if the number of rays is 16, have the number of columns in H(n) and PMAT to 64. This basically takes care of the zero padding.
3. Evaluate $H(n)$ on the 0th processor.
4. Pass the relevant rows to the various processors; eg., if there are 16 processors being used with total number of rows=64, pass 4 rows to each processor with processor 0 having rows 1-4, processor 1 having rows 5-8, processor 2 having rows 9-12 and so on for all the 64 rows. (This matrix is called PMAT in the program).
5. Since the same response function is used all through, this data is passed on to all the processors by a single broadcast message from processor 0 to all other processors.

   The remaining operations are carried out in parallel.

6. On each processor, evaluate FFT for each of the rows of PMAT. That is, find the FFT of each row-vector row by row on each processor. This step proceeds in parallel on all processors. At the end of this, each row of PMAT contains its corresponding FFT sequence.

7. Multiply each element in each row of PMAT with the corresponding element from H (the response function). This step again proceeds in parallel on all the processors. The result of this operation is stored in the same matrix PMAT. Also, multiply each element of PMAT with a Hamming window function to reduce the Gibbs phenomenon. (Any other window function could also be used. Not multiplying with a Hamming window function would lead to more errors in the final image).

8. Evaluate the IFFT of PMAT row-wise. This is done in a standard way - find the complex conjugate of each row of PMAT and find its FFT. Then find its complex conjugate again and normalize. This step proceeds in parallel on all the processors.

This finds us the filtered projections.

3.1.4 Parallel Computation of Backprojections

Mathematical computation of the backprojections is as given by -

$$f(x,y) = \frac{\pi}{K}\sum_{i=1}^{K} Q_{\theta_i}((x - x_{cen})/x_{cen})\cos\theta_i + ((y - y_{cen})/y_{cen})sin\theta_i) \tag{3.1}$$

where, x_{cen} and y_{cen} represent the center along the x and y co-ordinate axes, and, NP is the number of processors. Parallel implementation is carried out by splitting the above summation into smaller summations each of which can be computed in parallel. The results of these smaller summations are passed on to the zero'th processor which computes the total sum. The number of smaller summations into which the larger summation is split is equal to the number of processors. The parallel algorithm (including the linear interpolation) is as follows:

For x = 1,x_max

For y = 1,y_max

For i = 1,rows/NP

1. Choose x, y and a θ_i.

2. Find $t = ((x - x_{cen})/x_{cen})\cos(\theta_i) + ((y - y_{cen})/y_{cen})\sin(\theta_i)$. Find the two points closest to the above and linearly interpolate between them to find $Q_{\theta_i}(t)$ corresponding to the given x and y.

3. tmp_sum(x,y) = tmp_sum(x,y) + $Q_{\theta_i}(t)$

End for

End for

End for

4. Finally, Pass all the tmp_sum's to the zero'th processor to find the final sum.

3.2 Implementation on the Connection Machine

3.2.1 Architecture of the Connection Machine

The filtered back projection algorithm has been parallelized using the data parallel programming approach. The CM5 data network is well suited for data parallel programming as the machine has been designed for data parallel programming of large complex systems. The CM Fortran and C compilers support data parallel programming with a set of constructs [31] which allow users to program the machine for data parallelism. The CM5 runs the CMOST operating system which is based on UNIX with enhancements added to provide support for communication, time sharing and other activities important for running parallel

programs. The user normally uses one of the high level programming languages provided on the machine (CM Fortran, C^* or $^*LISP$) which are derivatives of standard programming languages, but with parallel constructs to exploit the data parallel programming approach. The Connection Machine is made up of a large number of processors, each with its own local memory. While programming the CM5, emphasis is on the use of large uniform data structures, such as arrays whose elements can be processed at once. Depending on the size of the arrays, each element can be assigned to one processor or many elements can be assigned to one processor. The elements however remain *distinct*, and each element can be considered to have its own virtual processor [31]. Unless we specify otherwise, arrays of the same size and shape will have identical layouts, and a statement like A = B + C would proceed in parallel on all the elements with A having the same size as B and C. For interconnected data structures (like the FFT where a butterfly type of interconnection is required) the serial algorithm has to be modified suitably for running optimally on the CM5. For fastest execution, the entire algorithm may have to be modified to achieve maximum efficiency. Parallelism for finding the FFT has been accomplished by using the FFT routines available on the CM5 (in the CMSSL library) as they have already been optimized for this machine. It may be noted that even faster FFT algorithms have been tested on the CM5. Swartzrauber [32] has tested a parallel FFT on the CM5 which is claimed to be faster than the CMSSL FFT. However, to keep matters simple, the Connection Machine Scientific Subroutine Library (CMSSL) [33] has been used for finding the FFT's and Inverse FFT's.

3.2.2 Parallel Computation of Filtered Projections

We have taken the data parallel approach for which the CM5 has been designed. Since in the serial algorithm, the filtered projections are found row-wise on the projection data, it was decided to find the FFT row-wise serial, but parallel within each row by using parallel FFT provided by CMSSL. Evaluation of the Hamming function and multiplying the Fourier transformed projection data by the response function and the Hamming window function has also been done in parallel within each row, but serially row after row. Hence

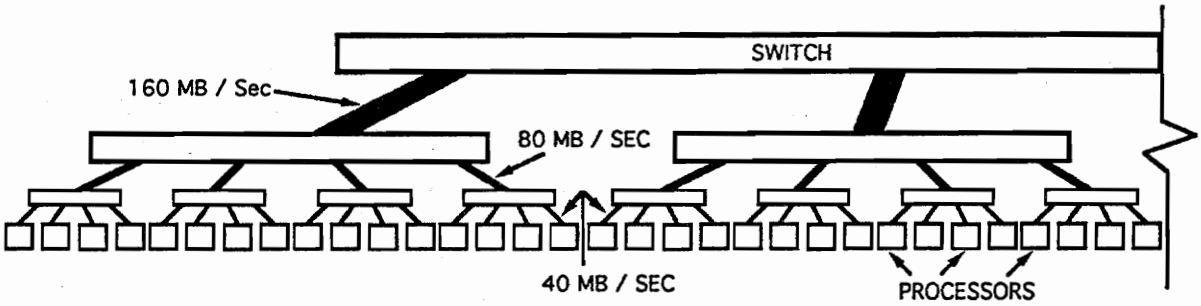

Figure 3.2: The Connection Machine (CM5) data network.

the algorithm for finding the filtered projections is:

```
For angle=1,K
For col = 1,max_cols
H2(col)=mat(angle,col) /* this assignment is also done in parallel for all elements in the
row */
End_for

    Find the FFT of H2 in parallel.

    Multiply H2 by the Hamming fn. and the filter fn.(ramp) in parallel.

    Find the IFFT of H2 in parallel.

    For col=1,max_cols
mat(angle,col)=H2(col) /* done in parallel */
End for
End for

    This finds us the filtered projections.
```

3.2.3 Parallel Computation of Backprojections

The back-projection operation requires an efficient parallel implementation of interpolation. From the serial program, it can be seen that for each value of x,y and angle, a distinct value of t has to be evaluated. Also, the values of low-bin (lb), high-bin (hb) and the mid-bin (mb) are also distinct for different values of x,y and angle. Hence, to achieve parallelism t,lb,hb and mb have been made into 3D matrices. This enables the future operations using these matrices to be done in parallel by using the parallel constructs available

on the CM5. The fractional values corresponding the mid-bin's and the low-bin's are stored in *frac* which is also a 3D matrix. Since *lb* and *hb* are references to the columns of the 2D matrix containing the filtered projections (*mat*), a double referencing is made, and the values corresponding to the low-bin and high-bin are stored in 3D matrices (*lbmat* and *hbmat*) . This allows us to carry out the linear interpolation in parallel (as is shown in the algorithm outlined below). Hence the parallel algorithm for back-projection is:

Each of the following steps is done in the order shown, but each step is executed in parallel:

For all x,y, and *angle* do in parallel each of the following steps:

1. Find $t(x, y, angle) = ((x - x_{cen})/x_{cen}) * \cos(\theta) + ((y - y_{cen})/y_{cen}) * \sin(\theta)$
2. Find mb(x,y,angle)=mp*t(x,y,angle)*correction + mp
3. Evaluate lb(x,y,angle) = truncate(mb(x,y,angle))
4. Evaluate frac(x,y,angle)=mb(x,y,angle)-lb(x,y,angle)
5. Evaluate hb(x,y,angle)=lb(x,y,angle)+1
6. Find lbmat(x,y,angle)=mat(angle,lb(x,y,angle))
7. Find hbmat(x,y,angle)=mat(angle,hb(x,y,angle))
8. For all x,y, and angle find the partial sums psum = lbmat + (hbmat-lbmat)*frac

   Note the double referencing being done in the evaluation of lbmat and hbmat.

   Finally, the densities can be found by summing over all the angles; this step is done in parallel by using the SUM function available on the CM5. This operation also proceeds in parallel, i.e.,

9. f(x,y) = sum of all the values of *psum* for all angles. This is done by summing the elements of *psum* over the *angle* (or third) dimension.

A detailed listing of all programs is available in the Appendix.

Chapter 4

Simulation Results

The simulations were performed for 1, 2, 4, 8 and 16 processors for the Intel Paragon and 32, 64, 128, 256 and 512 processors for the CM5. Plots of execution time, speed up and efficiency vrs. the number of processors have been made. The original images were of size 16x16, and the final images were 64x64 (due to zero padding). If the size of the original image is increased, then we can get better resolution in the final images. It should be noted that even though the final image is 64x64, it has information only from the original image which is 16x16.

The execution times were averaged over 8 runs for each case and are tabulated below (Tables 4.1, 4.2, 4.3). It can be seen from the Matlab simulation results (Table 4.1) that most of the floating point operations are spent in evaluating the back-projections - in fact the number of floating point operations for back-projections exceed those of the filtered projections by more than *14* times. Most of the cpu-time is therefore spent in evaluating the back-projections in both the parallel and the serial programs. Most of the speed up in the parallel programs is a result of the parallelization of the back-projections. Figures 4.1, 4.2 show the original and the reconstructed images for the Paragon and figures 4.6, 4.7 show the same for the Connection Machine (CM5). Figures 4.3, 4.4 and 4.5 show the trends in the execution time, speedup and efficiency for the Paragon. Figures 4.8, 4.9, 4.10, 4.11 show the speedup and efficiency trends in the case of the Connection Machine (CM5).

Speed up and efficiency has been calculated in the following manner:

$$N\ processor\ Speedup = \frac{total\ \ execution\ time\ on\ one\ processor}{total\ execution\ time\ on\ N\ processors} \tag{4.1}$$

$$Efficiency = \frac{Speed\ up}{Number\ of\ processors} \tag{4.2}$$

For the block efficiency curve of the CM5 in fig. 4.11, the following formula has been used:

$$Efficiency = \frac{Speed\ up}{Number\ of\ processors} * 32 \tag{4.3}$$

That is, the efficiencies have been normalized with respect to the efficiency of 32 processors on the CM5. This has been done because the minimum partition on the CM5 is a 32 node partition, and the other partitions must be a power of 2. Table 4.1 shows the simulation of the algorithm using Matlab on a Sun Sparc 10. It can be seen that the most of the floating point operations are consumed in the back-projection stage (about 93% of the total). Tables 4.2 and 4.3 show the simulations done on the Intel Paragon and CM5 respectively. The number of Million Floating Point Operations Per Second (MFLOPS) has been found by dividing the total number of floating point operations by the total time. It can be seen that the best simulation times are obtained with 8 processors on the Paragon and 256 processors on the CM5. Overall the Paragon seems to offer a better performance in terms of efficiency. Some preliminary simulations on the breakup of execution times between filtered projections and backprojections has shown that the CM5 takes longer for the filtered projections, and the Paragon takes longer for the backprojections. The times for the filtered projections on the Paragon showed a variation of as much as 40% in the 4 processor case. In the other cases, the variation was about 20% maximum. All the times shown here have been averaged over several runs (8 to 10) to get representative times for each case. On the CM5 the consistency was better and the times were consistent to about 10%.

It can be seen from the simulation results that in the case of CM5 the execution time is consistently dominated by the filtered projections, whereas in the case of the Paragon, the execution time is dominated by the back-projections.

Table 4.1: Matlab simulation results

Average Total Time(secs)	floating point operations for Filtered Projections	floating point operations for Back Projections
730.1	771,329	11,026,660

Table 4.2: Paragon simulation results

Number of Processors(NP)	Average Total Time	Average Time for Filtered Projections	Total MFLOPS
1	8.92	0.33	1.32
2	4.59	0.23	2.57
4	2.48	0.18	5.17
8	2.25	0.91	5.24
16	2.74	1.72	4.30

Table 4.3: CM5 simulation results

Number of Processors(NP)	Average Total Time	Average Time for Filtered Projections	Total MFLOPS
32	4.13	2.92	2.85
64	3.34	2.77	3.53
128	3.18	2.68	3.70
256	3.02	2.69	3.90
512	3.29	3.02	3.58

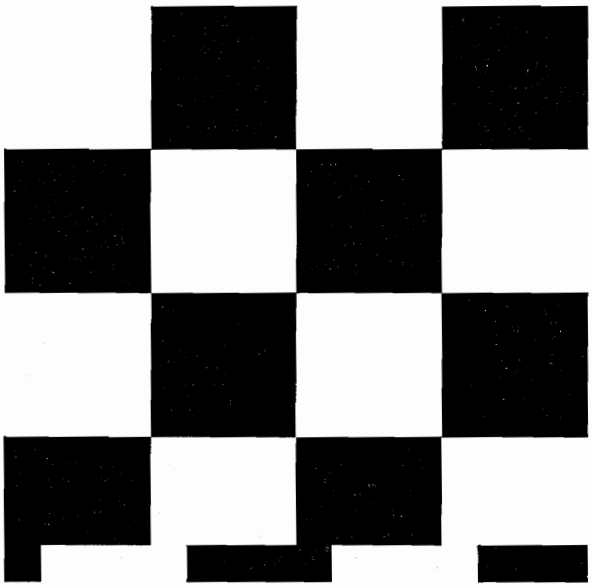

Figure 4.1: Original image (Paragon)

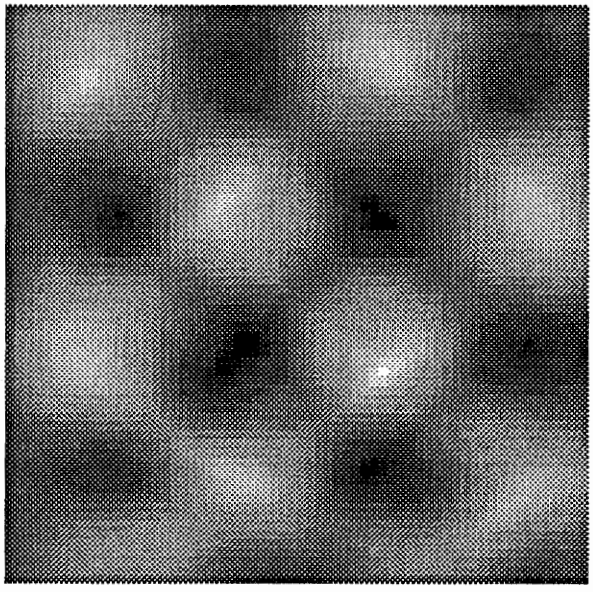

Figure 4.2: Reconstructed image (Paragon)

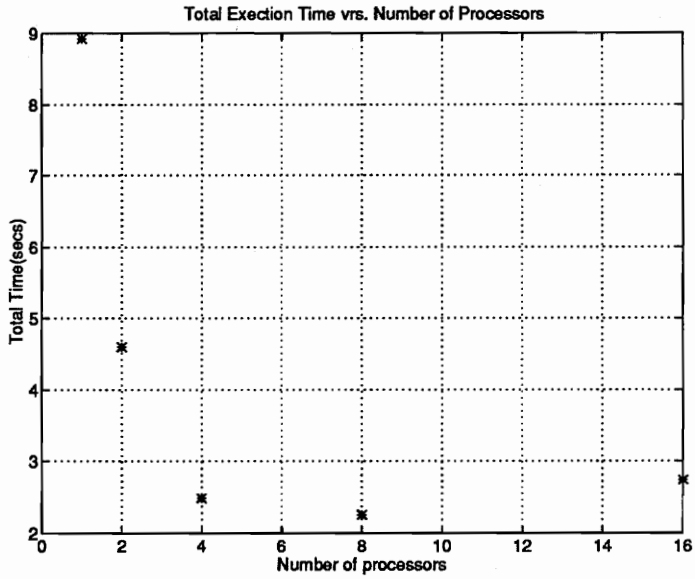

Figure 4.3: Execution Time (Paragon)

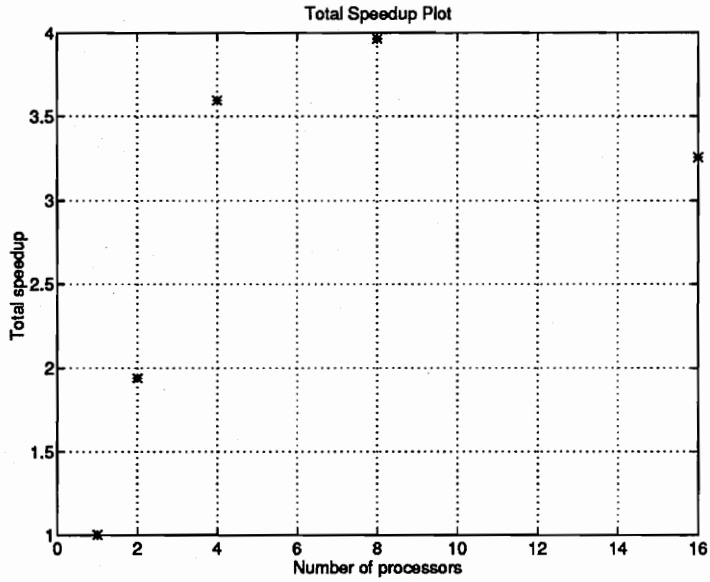

Figure 4.4: Paragon Speedup Curve

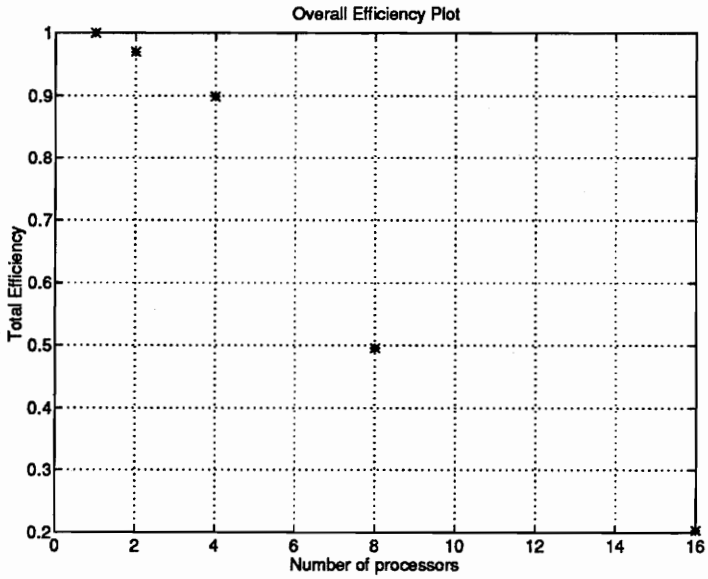

Figure 4.5: Paragon Efficiency Curve

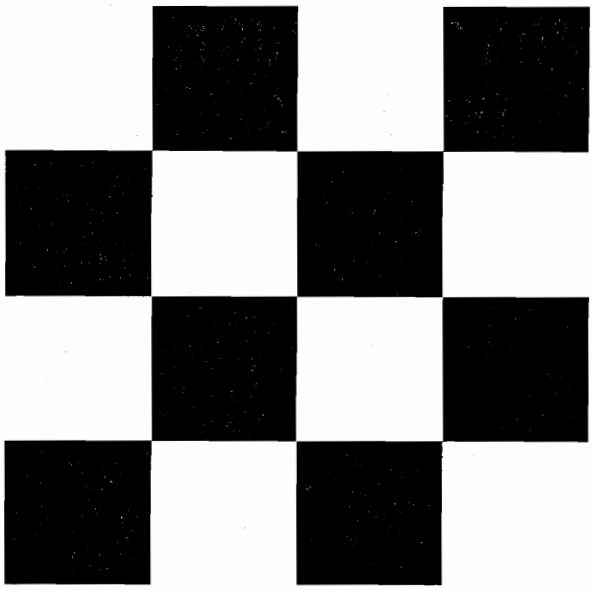

Figure 4.6: Original Image (CM5)

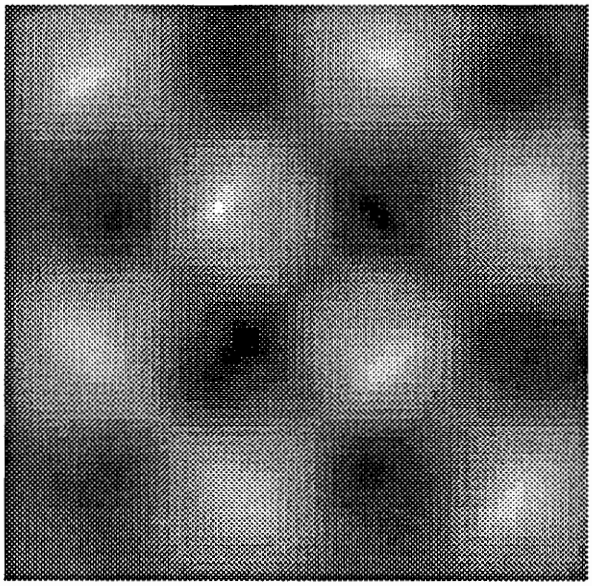

Figure 4.7: Reconstructed image (CM5)

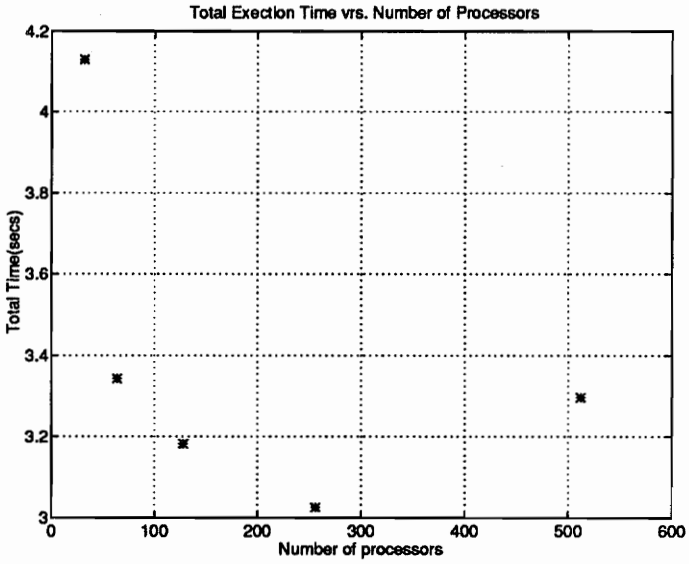

Figure 4.8: Execution Time (CM5)

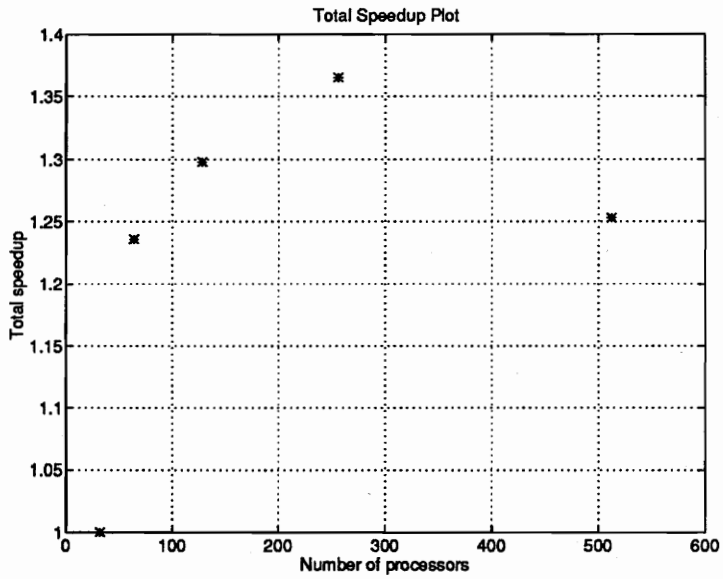

Figure 4.9: Speedup Curve (CM5)

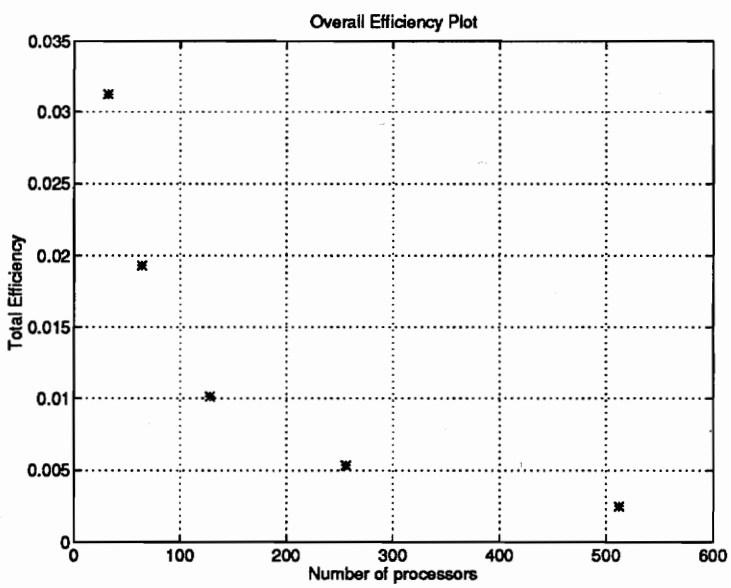

Figure 4.10: Efficiency Curve (CM5)

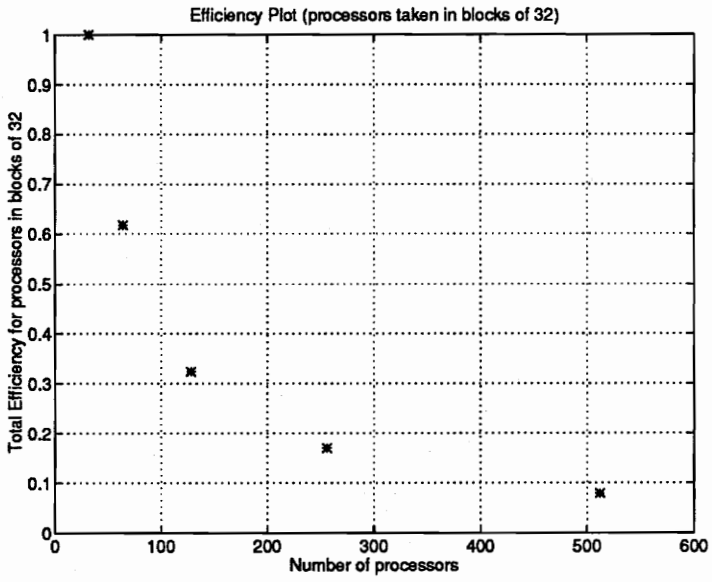

Figure 4.11: CM5 Efficiency Curve with processors taken in blocks of 32

Chapter 5

Conclusions And Scope For Further Work

5.1 Conclusions

The Filtered Backprojection Algorithm has been parallelized and run on a multiprocessor Intel Paragon and Connection Machine CM5. The algorithm has been analyzed for efficiency and speedup on both the Intel Paragon and the CM5. It was observed that maximum speedup is about 4 for the Intel Paragon and 1.36 for the CM5 (CM5 speedups are with respect to the 32 processor times). Overall, the results on the Intel Paragon (speed-up and efficiency) seem to be better than on the CM5. This could be due to the greater communication requirements between the processors of the CM5 as compared to the Intel Paragon. The larger number of processors on the CM5 also adds to the communication overhead. The maximum speed-up occurs for 8 processors on the Intel Paragon, and for 256 processors on the CM5. Speedup has been achieved mainly due to the parallelization of the back-projections, as most of the floating point operations are consumed in the back projection stage in the serial code. On the Paragon, in the back-projection stage, the splitting of the summation among different processors cannot be done indefinitely. At some point, the time taken to compute the partial summations will be less than the time for passing the results to the zeroth processor. At this point, increasing the number of processors will increase the execution time. On the CM5 the large memory requirements may be a disadvantage. The execution times obtained from the parallelization on the two machines indicate that at least in the 2D case real time performance can be expected.

From the simulations, it is seen that the CM5 algorithm spends most of the time on

calculating the filtered projections, and the Paragon in the calculating the back-projections. This difference in the two machines is primarily due to differences in the implementation.

5.2 Scope for Further Work

There are many problems which need to be addressed in the future. The parallel algorithms developed here can be easily extended to encompass fan beam and cone beam projections. Also since 3D reconstructions can be obtained by stacking 2D slices on the top of one another, the algorithms developed here can be run on several 2D slices to get 3D reconstructions. Another matter of interest is whether the CM5 algorithm can be improved in the evaluation of filtered projections so that further speed up can be obtained. These and other aspects need to be studied further, and any effort in this direction would be of value in acertaining real time performace for 3D reconstructions.

REFERENCES

[1] J. Radon, "Uber due bestimmung von funktionen durch ihre intergralwerte langs gewisser mannigfaltigkeiten (on the determination of functions from their integrals along certain manifolds," *Berichte Saechsische Akademie der Wissenschaften*, vol. 29, pp. 262 – 277, 1917.

[2] G. Hounsfield, "A method of an apparatus for examination of a body by radiation such as x-ray or gamma radiation," *Patent specifcation 1283915, The Patent Office*, 1972.

[3] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging.* IEEE, Inc., New York: IEEE Press, 1988.

[4] H. H. Barrett and W. Swindell, *Radialogical Imaging: The Theory of Image Formation, Detection and Processing.* New York: Academic Press, 1981.

[5] G. N. Ramanchandran and A. V. Lakshminarayanan, "Three dimensional reconstructions from radiographs and electron micrographs: Application of convolution instead of fourier transforms," *Proceedings of the National Academy of Sciences*, vol. 68, pp. 2236 – 2240, 1971.

[6] R. H. Bracewell and A. C. Riddle, "Inversion of fan beam scans in radio astronomy," *Astrophysics Journal*, vol. 150, pp. 427 – 434, 1967.

[7] A. V. Lakshminarayanan, "Reconstruction from divergent ray data," tech. rep., Dept. Computer Science, State University of New York at Buffalo, 1975.

[8] R. Gordon, R. Bender, and G. Herman, "Algebraic reconstruction techniques(ART) for three dimensional electron microscopy and X-ray photography," *Journal of Theoritical Biology*, vol. 36, pp. 105–117, 1970.

[9] R. Gordon, "A tutorial on ART(Algebraic Reconstruction Techniques)," *IEEE Transactions on Nuclear Science*, vol. NS-21, pp. 78–93, 1974.

[10] G. Herman and A. Naparstek, "Fast image reconstruction based on a Radon inversion formula approppriate for rapidly collected data," *SIAM Journal of Applied Mathematics*, vol. 33, pp. 511–533, 1976.

[11] G. Herman and A. Lent, "Iterative reconstruction algorithms," *Comput. Biol. Med.*, vol. 6, pp. 273–294, 1976.

REFERENCES

[12] T. Budinger and G. Gullberg, "Three-dimensional reconstruction in nuclear medicine emission imaging," *IEEE Transactions on Nuclear Science*, vol. NS-21, pp. 2–21, 1974.

[13] S. Wernecke and L. D. Addario, "Maximum entropy image reconstruction," *IEEE Transactions on Computing*, vol. C-26, pp. 351–364, 1977.

[14] L. Shepp and B. Logan, "The Fourier reconstruction of a head section," *IEEE Transactions on Nuclear Science*, vol. NS-21, pp. 21–43, 1974.

[15] E. Tanaka and T. A. Iinuma, "Correction functions for optimizing the reconstructed image in transverse section scan," *Phys. Med. Biol.*, vol. 20, pp. 789–798, 1975.

[16] S. Kenue and J. Greenleaf, "Efficient convolution kernels for computerized tomography," *Ultrasonic Imaging*, vol. 1, pp. 232–244, 1979.

[17] L.Wang, "Cross section reconstruction with fan beam scanning geometry," *IEEE Transactions on Computing*, vol. C-26, pp. 264–268, 1977.

[18] P. Dreike and D. Boyd, "Convolution reconstructions of fan-beam reconstructions," *Computer Graphics and Image Processing*, vol. 5, pp. 459–469, 1977.

[19] T. Peters and R. M. Lewitt, "Computed tomography with fan-beam geometry," *Journal of Computer Assisted Tomography*, vol. 1, pp. 429–436, 1977.

[20] C. Thompson and T. Peters, "A fractional address accumulator for fast backprojection," *IEEE Transactions on Nuclear Science*, vol. NS-28, no. 4, pp. 3648–3650, 1981.

[21] R. Hartz, D. Bristow, and N. Mullani, "A real time TOFPET slice backproject engine using dual Am 29116 microprocessors," *IEEE Transactions on Nuclear Science*, vol. NS-32, pp. 839–842, February 1985.

[22] Iskender Agi, P. J. Hurst, and K. W. Current, "An image processing IC for backprojection and spatial histogramming in a pipelined array," *IEEE Journal of Solid State Circuits*, vol. 28, no. 3, pp. 210 – 220, 1993.

[23] C. M. Chen, S. Y. Lee, and Z. H. Cho, "A parallel implementation of 3D CT image reconstruction on hypercube multiprocessor," *IEEE Transactions on Nuclear Science*, vol. 37, no. 3, pp. 1333 – 1346, 1990.

[24] Z. Cho, C. Chen, and S.-Y. Lee, "Incremental algorithms: fast back projection schemes for parallel beam geometries," *IEEE Transactions on Medical Imaging*, 1991.

[25] S. Barresi, D. Bollini, and A. D. Guerra, "Use of a transputer system for fast 3D image reconstruction in 3D PET," *IEEE Transactions on Nuclear Science*, vol. 37, no. 2, pp. 812–816, 1990.

REFERENCES

[26] W. Jones, L. Byars, and M. Casey, "Positron emission tomographic images and expectation maximization:A VLSI architecture for multiple iterations per second," *IEEE Transaction on Nuclear Science*, vol. 35, no. 1, pp. 620–624, 1988.

[27] J. Llacer and J. Meng, "Matrix based image reconstruction methods for tomography," *IEEE Transactions on Nuclear Science*, vol. NS-32, no. 1, pp. 855–864, 1985.

[28] M. Stella Atkins, Donald Murray, and R. Harrop, "Use of transputers in a 3D Positron Emission Tomograph," *IEEE Transactions on Medical Imaging*, vol. 10, no. 3, pp. 276 – 283, 1991.

[29] *A real time parallel processing data acquisition system*, December 1988.

[30] A. V. Oppenhiem and R. W. Schafer, *Discrete-time signal processing.* Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

[31] Thinking Machines Corporation, Cambridge, Massachusetts, *Connection Machine Fortran Reference Manual*, 1993.

[32] P. N. Swarztrauber, "Multiprocessor FFT's," *Parallel Computing*, vol. 5, pp. 197–210, 1987.

[33] Thinking Machines Corporation, Cambridge, Massachusetts, *Connection Machine Scientific Subroutine Library*, 1993.

Appendix A

Paragon Program Listing

```
      program filtered_bak
C*****************************************************************
C PROGRAM FOR CARRYING OUT TOMOGRAPHIC RECONSTRUCTION
C FOR 2d PROJECTION DATA FOR PARALLEL BEAM PROJECTIONS
C ON THE INTEL PARAGON.
C
C PROGRAMMER: RAMAN RAO
C DATE OF LAST REVISION: 1/2/95
C
C VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C BLACKSBURG, VA 24061
C*****************************************************************
C
C  u = # of rows per processor(=# of angles/nproc )
C  nproc = np = number of processors used
C np1 = np-1
C nnodes=number ofnodes (generated by the system)
C iam = my processor number
C ptype = process type
C szreal = number of bytes in type real in fortran
C power2 = length of the final image in the power of 2
C MAX = actual length of the final image
C rays = # of rays used in the forward projection
C rows=number of anglesa
C cols = number of detectors(rays)
C odim = object dimension
C rdim = reconstruction dimension (=xdim=ydim in this object only)
C xdim, ydim = size of the final image
C etime = elapsed time
C etime1-6 = elapsed times for intermediate stages of the program
C step = step size for generating the checker board
C blank = density allocated inside the object
C den = same as the above
```

APPENDIX A. PARAGON PROGRAM LISTING

```
C object = object to be projected and then backprojected
C r=xcos(theta)+ysin(theta) normalized to lie in the range (-1,1)
C pwr2=power of 2 (real), ipower2=int(pwr2)
C mat= initial projection matrix
C pmat = projection matrix on each processor. This is the same
C        as the matrix 'mat' except that the data contained in
C   'pmat' is different on each processor (contains data
C    pertinant to that processor)
C nbytes_pmat=number of bytes in the 'pmat' matrix
C H = row matrix of the filter/response function
C nbytes_h = number of bytes in the H matrix
C P = temporary row matrix for storing the rows of 'pmat'
C lb = low bin; hb = high bin; mb = mid bin
C i,j,k,iall,jall,ii1,l1,ii,ii2,ik: all are temporary variables
C for running the 'do loops'
C tmp_xy= 2D array for storing the partial summations in the
C          back-projection  stage
C sum_xy = 2D array for storing the summation for all (x,y)
C msg_xy = 2D array which is the same as tmp_xy but is a
C          message sent from some processor to the root
C          or zeroth processor
C f = the final matrix containing the reconstructed densities

      implicit none
      include 'fnx.h'
      integer max,angles,rays,xdim,ydim,u,nproc,power2,iix,ij
      integer odim,step,iproc
      parameter(MAX=64,angles=128,rays=16,xdim=64,ydim=64)
      parameter(odim=16,step=4)

  parameter(u=128,nproc=1)
  parameter(power2=6)
      complex xx,H,pmat,mat,P,tmp2
      real tau,theta,r,beta,frac,x_max,y_max,mb,mp,tw
      real delq,tmp_xy,T,sum,f,msg,pwr2,pi,x_cen,y_cen
      real tot_angle_sum,proc_angle_sum,sum_s,msg_s,const
      real proc_zero_ang_sum,sum_xy,msg_xy
      double precision stime,etime,etime1,etime2,etime3,etime4
      double precision etime5,etime6
      integer rows,cols,n1,k1,ptype,nnodes,iam,x,y,int_val
      integer iall,jall,i,j,k,nbytes_pmat,ii1,l1,ii,ii2,ik
```

```
      integer ipwr2,szreal,npp,np,nbytes_p,nbytes_h,lb,hb
      integer blank,den,i1,j1,np1
      real x_cen_o,y_cen_o,object
      dimension xx(16),H(MAX),mat(angles,rays),pmat(u,MAX)
      dimension P(MAX),T(30),f(xdim,ydim),sum(ydim),msg(ydim)
      dimension object(odim,odim)
      dimension proc_zero_ang_sum(MAX),proc_angle_sum(MAX)
      dimension tot_angle_sum(MAX),tmp_xy(xdim,ydim)
      dimension sum_xy(xdim,ydim),msg_xy(xdim,ydim)

C begin initializations
         etime=0.0
  etime1=0.0
  etime2=0.0
  etime3=0.0
  etime4=0.0
  etime5=0.0
  etime6=0.0
  stime=0.0
  PI = acos(-1.0)
c  stime = dclock()
c     write(*,*) 'PI=',PI
      np=nproc
      np1 = nproc-1
      write(*,*) 'np1=',np1
  szreal = 4
      rows = angles
      cols = rays
      tau = 1.0
      npp = rows/np
      nbytes_pmat = MAX*u*szreal*2
      nbytes_h = MAX*szreal*2
      x_max = real(xdim)
      y_max = real(ydim)
      x_cen = x_max/2.
      y_cen = y_max/2.
      x_cen_o=real(odim)/2.0
      y_cen_o=real(odim)/2.0

      mp = real(rays)/2.
      tw = 1./(1.15*sqrt(2.))
```

```
      const = PI/real(rows)
      write(*,*) 'const=',const
  ipwr2 = power2

C initialize system variables on each node
      iam = mynode()
      nnodes = numnodes()
      ptype = myptype()

C find the start time
  stime = dclock()

C init pmat in all processors to zero: this step takes care
C of the additional padding with zero's which is required later
       do iall=1,u
         do jall = 1,MAX
    pmat(iall,jall)=(0.,0.)
         enddo
       enddo
          etime1 = dclock()-stime
   write(*,*) 'etime1=',etime1

do iall=1,odim
  do jall=1,odim
    object(iall,jall)=0.
         enddo
        enddo

      if (iam. eq. 0) then
          write(*,*) 'Number of processors is',np
          write(*,*) 'angles=',rows,'rays=',cols

C generating the checker board (object to be reconstructed)
          blank =0
  do i=1,odim,step
    if (blank .eq. 0) then
      blank = 255
           else
              blank = 0
           endif
```

```
    do j=1,odim,step
      if (blank .eq. 0) then
blank = 255
             else
                blank = 0
             endif

       do i1=i,i+step
                do j1=j,j+step
                  object(i1,j1)=real(blank)
c     write(*,*) object(i1,j1),'i1=',i1,'j1=',j1
                enddo
              enddo

            enddo
           enddo

           etime2 = dclock()-stime
    write(*,*) 'etime2=',etime2

      write(*,*) 'finished generating the object'

C uncomment the following lines if you need to write the object to a file
c     open (unit = 19,file ='object.fil',status='old',form='formatted')
c     do iix=1,odim
c       do ii=1,odim
c         write(19,*) object(iix,ii)
c       enddo
c     enddo
c     close(19)
c     write(*,*) 'finished writing the object.fil'

C Evaluate the response function, find its fft
C and send this data to the other nodes (p. 72 of kak)
           call evalh2(H,1.0,cols)
           write(*,*) 'H evaluated'

C uncomment the following line if you need to write H to a file
c          open (unit = 15,file ='h.fil',status='old',form='formatted')
c          do i=1,MAX
c      write(15,*) 'i=',i,'H=',H(i)
```

```
c           enddo
c           close(15)

C broadcast the object and H to all the processors
            call csend(20,object,odim*odim*szreal,-1,ptype)
            call csend(20,H,nbytes_h,-1,ptype)

        else

C Give the corresponding receives to synchronize send's and receive's
            call crecv(20,object,odim*odim*szreal,0,ptype)
            call crecv(20,H,nbytes_h,0,ptype)

        end if

          etime3 = dclock()-stime
  write(*,*) 'etime3=',etime3

C forward project the object.
C This step directly generates the 'pmat' matrices
C for all the processors
C finding the forward projections in parallel on all the nodes

       do i=1,odim
           do j=1,odim
 do k=1,u

   theta = 2.0*real((u*iam)+k)*pi/real(angles)
   r = cos(theta)*((real(i)-x_cen_o)/x_cen_o)
     $                +sin(theta)*((real(j)-y_cen_o)/y_cen_o)
                 mb=(r*mp*tw)+mp
   lb=int(mb)
c    write(*,*) 'lb=',lb
   hb=lb+1
   frac=mb-lb
                 pmat(k,lb)=pmat(k,lb)+cmplx((1.0-frac)*object(i,j))
   pmat(k,hb)=pmat(k,hb)+cmplx(frac*object(i,j))

                enddo
            enddo
        enddo
```

```
 write(*,*) 'finished the forward projections'


          etime4 = dclock()-stime
   write(*,*) 'etime4=',etime4

C Now find the FFT of the input matrix row-wise on all the nodes
C this is done in parallel but row-wise serial on all the nodes
C The results of the FFT are again stored in the pmat matrix
       do i=1,u

C write a row of PMAT to P
         do j=1,MAX
           P(j) = PMAT(i,j)
         enddo

C find the FFT
         call diffft(P,ipwr2)

C write back P into PMAT
      do l1=1,MAX
               PMAT(i,l1) = P(l1)
             enddo
       enddo

C uncomment if you need to verify this step
c     open (unit = 18,file ='pmat1.fil',status='old'
c    $,form='formatted')
c     do i =1,np
c      if (iam .eq. i-1) then
c       do ii=1,u
c         write(18,*) 'iam=',iam,'row=',ii
c         write(18,*) (pmat(ii,ij), ij=1,MAX)
c       enddo
c      endif
c     enddo
c     close(18)

C multiply the two funs H and and pmat. Multiply here
C means scalar multiplication of equivalent elements in the
C same position (becomes convolution in the time domain).
```

```
C This part is executed in parallel on all the nodes
C ref( p. 75 kak). Multiplication with a Hamming function
C is also done right here

        do i=1,u
     do j=1,MAX
 PMAT(i,j) = PMAT(i,j) * H(j) *
     $ (0.54-0.46*cos((2.0*(real(j-1))*pi)/real(MAX)))
c       write(*,*) 'i=',i,'j=',j,'pmat=',PMAT(i,j)
            enddo
        enddo

C uncomment if you need to verify this step
c       open (unit = 17,file ='pmat2.fil',status='old',form='formatted')
c       do iix=1,nproc
c       call gsync()
c       if (iam .eq. iix-1) then
c       do ii=1,u
c       write(17,*) 'iam=',iam,'row=',ii
c       write(17,*) (pmat(ii,ij), ij=1,MAX)
c       enddo
c       endif
c       enddo
c       write(*,*) 'I finished the point to point multiply','iam=',iam
c       write(*,*) 'iam=',iam
c       close(17)

C find the row-wise IFFT of the new matrix
C IFFT is found by: cmplx conj. of the data row by row, finding the
C forward transform, again cmplx conj. and then multiply by the
C scale factor. This step proceeds in parallel on all the
C nodes(ref. any d.s.p. book or d.s.p. by Oppenheim and Schafer.

C for all rows on the processor do:
       do i=1,u

C move a row from PMAT to P
   do j=1,MAX
      P(j) = PMAT(i,j)
          enddo
```

```
C find the complex conj. of each element
          do ij=1,MAX
          tmp2 = P(ij)
           P(ij) = conjg(tmp2)
          enddo

C find the FFT
   call diffft(P,ipwr2)

C find the complex conj. of each element
           do ij=1,MAX
             tmp2 = P(ij)
           P(ij) = conjg(tmp2)
           enddo

C normalize by the scale factor
            do  ik =1,MAX
tmp2= P(ik)
P(ik) = (tmp2)/real(MAX)
            enddo

C put it bak in the original matrix
            do j=1,MAX
      PMAT(i,j) = P(j)
            enddo

         enddo

        write(*,*) 'Finished calculating the filtered vals'

C synchronize all processors: This is done to time the filtered
C projections
            call gsync()
              etime5 = dclock() - stime
write(*,*) 'etime5=',etime5
        etime = etime1 + (etime5-etime4)

            if (iam .eq. 0) then
        write(*,*) 'time for calculating filtered projections is',etime
 write(*,*) 'iam=',iam
            endif
```

```
C uncomment if you need to verify results at this stage
c       open (unit = 13,file ='pmat3.fil',status='old',form='formatted')
c       do iix=1,nproc
c         call gsync()
c         if (iam .eq. iix-1) then
c            do ii=1,u
c            write(13,*) 'iam=',iam,'row=',ii
c            write(13,*) (PMAT(ii,ij), ij=1,MAX)
c      enddo
c         endif
c       enddo
c       close(13)

c         do i=1,u
c           theta = real((u*iam)+i)*PI/real(angles)
c           write(*,*) 'theta=',theta,'iam=',iam
c         enddo

C Backprojection begins here
C
C initialize
      do x=1,xdim
 do y=1,ydim
   tmp_xy(x,y)=0.0
   msg_xy(x,y)=0.0
   sum_xy(x,y)=0.0
        enddo
      enddo

C now carrying out the summation in the backprojection stage
C (ref. p.67 kak. The summation on p.67 is split into smaller
C summations on each processor. Then these smaller summations
C are transferred to the zeroth processor for finding the
C total sum.
C
C finds the tmp_xy's for each x and y

C for all x and y do:
      do x = 1,xdim
         do y= 1,ydim
```

```
C fo all rows on each processor do:
            do i= 1,u
C theta evaluated: Note 2.0*PI  may not be required
C
C Just PI will suffice as the multiplier
               theta = 2.0*PI*real((u*iam)+i)/real(angles)

C find the value of 't'(ref. p49 kak)
               r = cos(theta)*((x-x_cen)/x_cen)
     $               + sin(theta)*((y-y_cen)/y_cen)
C
C find the mid bin, low bin and high bin
               mb= (r*mp*tw) + mp
               lb = int(mb)
               hb = int(mb)+1
C
C find the fractional value of the mid bin
               frac = mb-lb
               tmp_xy(x,y)=tmp_xy(x,y)+
     $                    real(PMAT(i,lb))*(1.0-frac) +
     $                         frac*real(pmat(i,hb))
c  write(*,*) 'mb=',mb,'lb=',lb,'hb=',hb,'frac=',frac
     enddo
          enddo
       enddo

C transfer all the tmp_xy's to the zeroth processor
C and add them all up.

          if (iam .ne. 0) then
               call csend(20,tmp_xy,xdim*ydim*szreal,0,ptype)
          else

C if iam the zeoro'th processor:
C initialize the sum to zero
       do x=1,xdim
                do y=1,ydim
                  sum_xy(x,y) = 0.0
                enddo
              enddo
```

```
C for all the processors on the system other than zero do:
              do ii1 = 1,np1
C
C receive the tmp_xy sent earlier
                call crecv(20,msg_xy,xdim*ydim*szreal,ii1,ptype)
C
C add them all up as they are received
  do x=1,xdim
    do y=1,ydim
                    sum_xy(x,y)=sum_xy(x,y)+msg_xy(x,y)
                  enddo
                enddo

              enddo

C find the density matrix
      do x=1,xdim
               do y=1,ydim
                 f(x,y)=const*(sum_xy(x,y)+tmp_xy(x,y))
               enddo
              enddo

C endif for the zero'th processor
           endif

C synchronize all the processors for getting the timing information
       call gsync()

       if (iam .eq. 0) then
         etime6 = dclock() - stime
  write(*,*) 'etime6=',etime6
      etime = etime1+(etime5-etime4)+(etime6-etime5)
         write(*,*) 'total elapsed time=',etime,'iam=',iam
C
C write the densities to a file
      open (unit = 11,file ='f.fil',status='old',
     $   form='formatted')
      write(11,*), ((f(i,j),j=1,ydim),i=1,xdim)
      close(11)
       endif
```

```
C end of main program
 stop
 end

C finds the FFT. The method used is 'decimation in time'
          SUBROUTINE DIFFFT(X,NU)
          COMPLEX X(1024),U,W,T
          N=2**NU
          PI=3.14159265358979


          DO 20 L=1,NU
          LE=2**(NU+1-L)
          LE1=LE/2
  U=(1.,0.)
   W=CMPLX(COS(PI/FLOAT(LE1)),-SIN(PI/FLOAT(LE1)))
  DO 20 J=1,LE1
      DO 10 I=J,N,LE
    IP=I+LE1
    T=X(I)+X(IP)
    X(IP)=(X(I)-X(IP))*U
   10                X(I)=T
   20            U=U*W

   NV2=N/2
   NM1=N-1
   J=1
   DO 30 I=1,NM1
  IF (I .GE. J) GO TO 25
 T=X(J)
 X(J)=X(I)
 X(I)=T
   25            K=NV2
   26            IF (K .GE. J) GO TO 30
 J=J-K
 K=K/2
                 GO TO 26
   30      J=J+K

    RETURN
```

```
    END

C evaluates the filter (or response function) in the
C frequency domain.
      subroutine evalh2(H,tau,N)
      real tau, PI
      integer N,j,i,k,ii,MAX,HMAX
      parameter(MAX=64,HMAX=32)
      complex H(MAX),a1(HMAX)
      PI = acos(-1.0)
      do j=1,MAX
        H(j) = (0.,0.)
      enddo

      do i=1,HMAX
        a1(i) = real(i)
      enddo

      do i =1,MAX
        if (i .le. HMAX) then
          H(i) = real(i)
        else
          H(i) = real(MAX)+1.0 -real(i)
        endif
      enddo

C Optional: can multiply with a Hamming window function here.
C Not required really, but can be used to get better(?) images.
C Note that the filter function C changes from a ramp on using
C this Hamming multiply.
c     do i = 1,MAX
c       H(i) = H(i)*(.54 + .46*cos(2.0*pi*real(i)/(real(MAX))))
c     enddo

      return
      end
```

Appendix B
CM5 Program Listing

```
C ******************************************************************
C PROGRAM FOR TOMOGRAPHIC RECONSTRUCTON OF PARALLEL BEAM DATA
C ON THE CONNECTION MACHINE CM5.
C
C PROGRAMMER: RAMAN RAO
C DATE OF LAST REVISION: 1/2/95
C VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C BLACKSBURG, VA 24061
C******************************************************************


C This program carries out CT reconstruction on the CM5
C for parallel beam projection data.
C
C odim=object dimension
C rdim=reconstruction dimension
C object=object (2D) to be forward projected
C lb1 = stores low bins for the forward projections
C hb1=stores high bins for the forward-projections
C mb1=stores the mid bin values for the forward projections
C frac1=stores the fractional part of the mid bin for forward projections
C step = the width of each block in the checker board
C angles=number of angles at which the forward projections
C are performed
C mat = projection matrix (proj. data is stored here)
C lb2 = stores low bins for the back-projections
C hb2=stores high bins for the back-projections
C mb2=stores the mid bin values for the back-projections
C frac2=stores the fractional part of the mid bin for back projections
C f = final density matrix for the reconstructed object
C Ham=hamming window function is generated and stored here
C H2=temporary row vector for storing a row
C lbmat=stores the filtered proj. values corresponding to the low bin
C hbmat=stores the filtred proj. values corresponding to the high bin
```

```
C psum=stores the interpolated values corresponding to the mid-bins
C xcen1, ycen1= center of the object dimension
C xcen2, ycen2=center of the reconstructed dimension
C xmax,ymax = maximum size of the z and y dimensions
C mp1= mid point of the rays
C tw = adjustment for keeping the mb inside the object(adjust until
C the program gives no error
C H1 stores the ramp function (response function)
C evalh = subroutine which generates the response function
C theta= angles which are used in the projection
C The CMF$ commands enable the respective arrays to be executed
C in parallel.

C The data of dimension odimxodim is reconstructed to rdimxrdim.
C In this process, zero padding is done to fill the frequency values
C for dimensions greater than odim (p. 75 kak) to reduce the dishing
C and dc artifacts(fig. 3.17,p.76 kak)
C

      program  filtered_bak
      implicit none
C include the cmssl routines
      include '/usr/include/cm/cmssl-cmf.h'
C include the timer routines
      include '/usr/include/cm/timer-fort.h'

      integer odim,rdim,step
      parameter (odim=16,rdim=64,step=4,angles=128)
      integer rows,cols,i,j,k,setup_id,ier,n1,rays,blank,i1,icen
      integer hb1(1:odim,1:odim,1:128),lb1(1:odim,1:odim,1:128),j1
CMF$  LAYOUT  hb1(:news,:news,:news),lb1(:news,:news,:news)
      integer hb2(1:rdim,1:rdim,1:128),lb2(1:rdim,1:rdim,1:128)
CMF$  LAYOUT  hb2(:news,:news,:news),lb2(:news,:news,:news)
      complex mat(1:128,1:rdim),H1(1:rdim),H2(1:rdim)
CMF$  LAYOUT  mat(:news,:news),H1(:news),H2(:news)
      real tau1,tw,xcen1,xmax,ycen1,ymax,f(1:rdim,1:rdim),Ham(1:rdim)
CMF$  LAYOUT f(:news,:news),Ham(1:news)
      real theta(1:128),frac1(1:odim,1:odim,1:128),
     $  mb1(1:odim,1:odim,1:128)
CMF$  LAYOUT theta(:news),frac1(:news,:news,:news)
CMF$  LAYOUT  mb1(:news,:news,:news)
```

```
      real frac2(1:rdim,1:rdim,1:128),mb2(1:rdim,1:rdim,1:128)
CMF$  LAYOUT  frac2(:news,:news,:news),mb2(:news,:news,:news)
      real PI,psum(1:rdim,1:rdim,1:128),p1(1:odim,1:odim,1:128)
CMF$  LAYOUT psum(:news,:news,:news),p1(:news,:news,:news)
      real p2(1:rdim,1:rdim,1:128)
CMF$  LAYOUT p2(:news,:news,:news)
      real hbmat(1:rdim,1:rdim,1:128),lbmat(1:rdim,1:rdim,1:128),const
CMF$  LAYOUT hbmat(:news,:news,:news), lbmat(:news,:news,:news)
      real object(1:odim,1:odim),den,frac
CMF$  LAYOUT object(:news,:news)
      real xcen2,ycen2,mp1,mp2
      integer lb,hb,timer1

      timer1=0
C clear the timer
      call CM_timer_clear(timer1)

C start the timer
      call CM_timer_start(timer1)

      xcen1=real(odim)/2.
      ycen1=real(odim)/2.
      xcen2 = real(rdim)/2.
      ycen2 = real(rdim)/2.

      tw = 1.0/(1.15 * sqrt(2.))
      rays = odim
      mp1=real(rays)/2.
      mp2 = real(rdim)/2.

      n1=6
C rows = # of angles
      rows=128
      cols=rdim

      PI = acos(-1.0)
      const = PI/real(rows)

C initialize the mat, H1 (ramp function), and object
      forall(i=1:rows,j=1:cols) mat(i,j) = (0.0,0.0)
      forall(j=1:cols) H1(j) = (0.0,0.0)
```

```
      forall(i=1:odim,j=1:odim) object(i,j) =0.0
       print *, "mat and H1 init completed"
       call CM_timer_stop(timer1)

C this routine projects the data . The data is 128 128 x odim rows,
C and the data is supposed to look like a checker board.
C making the checker board

      blank =0
      do i =1,odim,step
 if (blank .eq. 0) then
   blank = 255
        else
   blank = 0
        endif

 do j=1,odim,step

   if (blank .eq. 0) then
     blank = 255
          else
            blank = 0
          endif

 forall(i1=i:i+step,j1=j:j+step)
     $  object(i1,j1) = real(blank)

        enddo
       enddo

        call CM_timer_start(timer1)

C Begins the forward projecton process. Projections are found by adding
C the densities along parallel rays (since this is parallel
C beam tomography)

C finds theta for (in radians) for all the steps
      theta = real([1:128])*PI/128.
       print *, 'theta evaluated'

C stop timer
```

```
      call CM_timer_stop(timer1)

C finds the value of (xcos(theta)+ysin(theta)) for all the
C angles and all x,y. The values below are normalized to
C make the forward projection space to lie in the space (-1,1)
      forall(i=1:odim,j=1:odim,k=1:128)
     $   p1(i,j,k) = ((real(i)-xcen1)/xcen1)*cos(theta(k))
     $ + ((real(j)-ycen1)/ycen1)*sin(theta(k))
       print *, 'p1 evaluated'

C find the mid-bin of the forward projections
      forall(i=1:odim,j=1:odim,k=1:128)
     $ mb1(i,j,k) = (p1(i,j,k)*mp1*tw) + mp1
C find the low bins
      lb1 = int(mb1)
       print *, 'lb1 evaluated'

C find the fractional values
      frac1 = mb1-real(lb1)
      print *, 'frac1 evaluated'

C find the high bins
      forall(i=1:odim,j=1:odim,k=1:128) hb1(i,j,k) = lb1(i,j,k) + 1
      print *, 'hb1 evaluated'

C following is executed serially: not timed as these carry out the forward
C projections
      do i=1,odim
  do j=1,odim
     do k=1,128
       lb = lb1(i,j,k)
hb = hb1(i,j,k)
frac = frac1(i,j,k)
den = object(i,j)
mat(k,lb) = mat(k,lb)+cmplx((1.0-frac)*den)
mat(k,hb) = mat(k,hb)+cmplx(frac*den)
              enddo
           enddo
       enddo

      call  CM_timer_start(timer1)
```

```
C evaluate the response function(ref.  p72 kak)
      call evalh(H1,tau1,n1)
      print *, 'H matrix evaluation completed'

C Find the FFT of the projection matrix row by row.
C The parallel FFT on the CM5 is used to find the fourier
C transforms row by row.
      do i=1,rows
  forall(j=1:cols) H2(j) = mat(i,j)

C setup the FFT variables inside the CM5
        setup_id = fft_setup(H2,'CTOC',ier)
        call fft(H2,'CTOC',CMSSL_f_xform,setup_id,ier)
 if (ier .ne. 0) then
   write(*,*) 'error in fft','i=',i,'ier=',ier
        endif

C deallocate the FFT setup variables
        call deallocate_fft_setup( setup_id)
  forall(j=1:cols) mat(i,j) = H2(j)
      enddo
        print *, 'forward transform of MAT completed'

C point to point multiply of the two fft's-the fft of the
C proj. matrix and the response function.
C Multiplies each row in parallel one row at a time
C Also multiplied point to point by the Hamming window function to
C reduce the gibbs phenomenon (p.75 kak)
      forall(i = 1:cols)
     $ Ham(i)=(0.54-0.46*cos((2.0*(i-1)*PI)/real(cols)))
      do i=1,128
       mat(i,:) = mat(i,:) * h1 * cmplx(Ham)
      enddo
       print *, 'point to point multiply completed'

C find the inverse FFT
C Inverse FFT is found by finding the complex conj of the
C matrrix row by row, finding the forward FFT , again finding
C the complex conj. and then scaling(ref. oppenheim and schafer
C : digital signal processing, or any other dsp book)
```

```
      do i =1,rows
  forall(j=1:cols) H2(j) = mat(i,j)
         setup_id = fft_setup(H2,'CTOC',ier)
         call fft(H2,'CTOC',CMSSL_i_xform,setup_id,ier)
 if (ier .ne. 0) then
   write(*,*) 'error in ifft','i=',i,'ier=',ier
        endif
         call deallocate_fft_setup( setup_id)
  forall(j=1:cols) mat(i,j) = H2(j)
      enddo
         print *, 'inverse fft completed'
      call CM_timer_stop(timer1)
      write(*,*) 'time for calculating filtered projections'
      call CM_timer_print(timer1)
      call CM_timer_start(timer1)

C Interpolate to find the actual values. The interpolated
C values are stored in the  variable 'psum'
C The code below actually does the backprojection, but some
C of the code which has been executed earlier is also
C timed as it is required for back-projection.
C Finds  the value of (xcos(theta)+ysin(theta)) with x and y
C normalized to lie in the space (-1,1)
      forall(i=1:rdim,j=1:rdim,k=1:128)
     $   p2(i,j,k) = ((real(i)-xcen2)/xcen2)*cos(theta(k))
     $                + ((real(j)-ycen2)/ycen2)*sin(theta(k))
       print *, 'p2 evaluated'

C finds the mid bins in parallel
      forall(i=1:rdim,j=1:rdim,k=1:128)
     $ mb2(i,j,k) = (p2(i,j,k)*mp1*tw) + mp1

C finds the low bins in parallel
      lb2 = int(mb2)
       print *, 'lb evaluated'

C finds the fractional values of the mid bins
      frac2 = mb2-real(lb2)
      print *, 'frac2 evaluated'

C finds the high bins from the low bins
```

```
      forall(i=1:rdim,j=1:rdim,k=1:128) hb2(i,j,k) = lb2(i,j,k) + 1
      print *, 'hb2 evaluated'

C finds the filtered values corresponding to the low bins
      forall(i=1:rdim,j=1:rdim,k=1:128)
     $ lbmat(i,j,k) = real(mat(k,lb2(i,j,k)))
       print *, 'lbmat2 evaluated'

C finds the filtered values corresponding to the high bins
      forall(i=1:rdim,j=1:rdim,k=1:128)
     $ hbmat(i,j,k) = real(mat(k,hb2(i,j,k)))
       print *, 'hbmat2 evaluated'

C find the interpolated values corresponding to the fractional
C values just evaluated
      psum = ((1.-frac2)*lbmat) + (frac2*hbmat)
      print *, 'psum evaluated'

C finds the total summation over all the angles by summing over
C all the angles
        f = SUM(psum,dim=3)
 call  CM_timer_stop(timer1)
 call CM_timer_print(timer1)
 print *,'f matrix evaluated'

C write the 'f' matrix to a file
       open(unit=11,file='cm5_f.fil',status='old',form='formatted')
       write(11,*), ((f(i,j),j=1,rdim),i=1,rdim)
close(11)



      end

C finds the response function directly in the frequency
C domain. (Just a ramp function over the reconstruction space)
      subroutine evalh(H,tau,N)
      real tau, PI
      integer N,j,i,k,ii,rdim
      parameter (rdim=64)
      complex H(1:rdim),a1(32)
```

APPENDIX B. CM5 PROGRAM LISTING

```
CMF$    LAYOUT h(:news),a1(:news)


       PI = acos(-1.0)

forall(j=1:rdim) H(j) = (0.,0.)

      forall (i=1:32)
     $  a1(i) = real(i)

      do i =1,rdim
        if (i .le. 32) then
     H(i) = real(i)
        else
     H(i) = 65.-real(i)
        endif
      enddo

C can multipy with a Hamming function if necessary
c      do i = 1,rdim
c        H(i) = H(i)*(.54 + .46*cos(2*pi*real(i)/real(rdim)))
c      enddo



return
end
```

Appendix C
Matlab Program Listing

```
% This is matlab code which does the ct reconstruction for
% parallel beam x-ray tomography. The code has been written to verify
% the correctness of the fortran algorithm.
% The program will give an image pertaining to the projection
% data by making gray scaled image of the final data.
% It takes about 15 minutes on a Sun Sparc 10 for an image of 64x64 pixels.

clear; tau=1.0;  rays=16;  angles=128;  flops(0);

% In this program the variable, tau is set to unity, but is defined
% in more general terms on pgs. 71-72 (Fig 3.14) "Kak and Slaney".

mp=rays/2.0;
x_min=1
x_max=rays;
x_cen=((x_max)/2.0)
y_min=1
y_max=rays;
y_cen=((y_max)/2.0)
x_max2 = 64;
y_max2= 64;
x_cen2 = x_max2/2;
y_cen2 = y_max2/2;
pi = acos(-1.0)
disp('Calculations in progress..........');

%initialize the object, the original pattern, to zeros
disp('initlalizing the object');
 for i = 1:x_max,
   for j = 1:y_max,
     object(i,j) = 0;
   end;
```

```
 end;

% initalize the projection matrix to zero:
% simulated parallel xray projections [eqn(3) pg. 50]
disp('initlalizing the projection matrix');
 for i = 1:angles,
   for j = 1:rays,
     A(i,j) = 0;
   end;
 end;

nf1=flops;

% This routine generates data of the original object: (x_max, y_max),
% The pattern is supposed to look like a checker board.
% Make a checker board pattern, see Fig. 4.6 pg. 36 of this report.
  blank =0;
    for i =1:4:x_max,
          if blank == 0 blank = 255;
          else
            blank = 0;
          end;

      for j=1:4:y_max,
              if blank ==0 blank = 255;
              else
                blank = 0;
              end;

                  for i1=i:i+4,
                      for j1=j:j+4,
                         object(i1,j1) = blank;
                      end;
                  end;

      end;
    end;

nf2=flops;

% Chris Henze's ramp function: (Biology)
```

```
disp('generating the ramp');
a1 = linspace(0,32,32);              % create linear ramp, pg 168 MATLAB
hf = [a1 fliplr(a1(1:32))];           % flip left-to-right, pg 77 MATLAB
H = hf .* hamming(64)';             % Hamming funct, pg 168 MATLAB

nf3=flops;

disp('Using Hamming Window N =64');
% projecting the data points. The projections are
% 64 angles X 16 rays per angle.

disp('Projecting the object');
% The factor=1.15 in the denominator requires some explanation.
% Ideally this should be just 1/sqrt(2).  This factor,
% 1/sqrt(2), is the worst case at a rotation of 45 degrees where
% the object always fits inside a square whose edge dimension is
% the largest dimension of the original object.  Pragmatically,
% this factor 1.15 has been put in the denominator so that rays
% always fit within the object.  When the rays fall outside of the
% object, although the result should be near zero, a numerical
% error results.  If 1.15 is changed to 1.0 this program will not
% run.  An alternate technique may work if the uneven object fits
% into a circle instead of a square, where the largest object
% dimension becomes the diameter of the circle.

tw=1/(1.15*sqrt(2));

% Calculate the forward projections: Kak & Slaney pgs. 49-56
% This is where the CM5 does not parallelize well, but we can
% use the Paragon to parallelize this section by assigning
% rows(angles) to each cpu(node).

  for x=x_min:x_max,
     for y=y_min:y_max,

                sum=0.0;

                for i=1:angles,

                    theta = (i-1)*pi/angles;
r = cos(theta)*((x-x_cen)/(x_cen))+sin(theta)*((y-y_cen)/(y_cen));
```

```
                        mb = (r*mp*tw)+mp;
                        lb = floor(mb);hb=ceil(mb);frac=mb-lb;
                    A(i,lb) = A(i,lb) + ((1-frac)*object(x,y));
                    A(i,hb) = A(i,hb) + (frac*object(x,y));
                end;
      end;
   end;

disp('forward projections completed');
nf4=flops;

% Calculate the 1D FFT, Step 1, pg 15 Report, also eqn(2.4)
for i=1:angles,
    for j=1:rays,
      p1(j)=A(i,j);
    end;

% pad with zeros to the right of the matrix p1
      p = [p1 zeros(1,48)];  % Extend p1(16) to 64 by padding with zeros

      a2 = fft(p);

% The next three lines could be eliminated but they
% were included here to view intermediate values
%   for j=1:length(a2),
%     A2(i,j)=a2(j);
%   end;

% Recall, H, Hamming operation, see the inside integral of eqn(33) pg 64
% of Kak & Slaney.  Also see this same operation on the bottom line of
% eqn(69) on pg 75, H = [FFT h (n t) with ZP]xsmoothing-window}.
  dtime= fft(p) .* H;

% Again the next three lines could be eliminated but they
% were included here to view intermediate values
%  a3 = dtime;
%  for j=1:length(a3),
%    A3(i,j)=a3(j);
%  end;

% The left operation of eqn(69)
```

```
  d = ifft(dtime);
  for j=1:length(d),
    c(i,j)=d(j);
  end;

% This is the end of the loop on angles
end;

disp('Filtered Projections Completed');
nf5 = flops;

disp('backprojecting...');
% This is the final step of backprojecting the results into array f(x,y).
% This is the summation of step 4 eqn(2.21) on pg 15 in this report,
% also eqn(45) on pg 67 of Kak & Slaney.
% This operation is similar to the forward projection
% except now we have to sum (this sum can be divided over cpu's) instead of
% the arbitrary "random" accumulation of values of the forward projection.
% Comment:  This part is important for both the CM5 & the Paragon: each
% of these machines parallelize this sum differently but this is the most
% important part of the total time on either machine and represent the
% section that benefits the most from the parallelization on the CM5 or
% the Paragon.

  for x=x_min:x_max2,
   for y=y_min:y_max2,

         sum=0.0;
         for i=1:angles,
             theta = (i-1)*pi/angles;
r = cos(theta)*((x-x_cen2)/(x_cen2))+sin(theta)*((y-y_cen2)/(y_cen2));
                 mb = (r*mp*tw)+mp;
                 lb = floor(mb);hb=ceil(mb);frac=mb-lb;
             sum = sum + real(c(i,lb))*(1-frac) + frac*real(c(i,hb));
          end;

     f(x,y)=sum;

   end;
  end;
disp('backward projection completed');
```

```
disp('store reconstructed image to f.dat');
save f.dat f /ascii

disp('DONE')

%fid = fopen('mag.gray','wb');
%fwrite(fid,fgray,'real');
%nf6=flops;
%save ct26_0117.mat;
%quit;
```