A PARALLEL IMPLEMENTATION OF 3-D CT IMAGE RECONSTRUCTION ON HYPERCUBE MULTIPROCESSOR†

C. M. Chen and S.-Y. Lee
School of Electrical Engineering
Cornell University
Ithaca, NY 14853

Z. H. Cho*
Department of Radiological Science
University of California at Irvine
Irvine, CA 92717

Abstract

In this paper, we describe how image reconstruction in Computerized Tomography (CT) can be parallelized on a message-passing multiprocessor. In particular, the results obtained from parallel implementation of 3-D CT image reconstruction for parallel beam geometries on the Intel hypercube, iPSC/2, are presented. A two stage pipelining approach is employed for filtering (convolution) and backprojection. The conventional sequential convolution algorithm is modified such that the symmetry of the filter kernel is fully utilized for parallelization. In the backprojection stage, the 3-D *Incremental* algorithm, our recently developed backprojection scheme which is shown to be faster than conventional algorithm, is parallelized. The speed-up, defined as (sequential processing time)/(parallel processing time), ranging from 5 to 27, and the efficiency, defined as (speed-up)/(the number of processing elements), ranging from 60% to 92%, have been achieved, depending on the size of image and the number of processing elements employed.

1. Introduction

Computerized Tomography has played an important role in medicine since the early 1970's and is expected to become an inevitable diagnostic technique in the future. However, the long time required to obtain an image has been one of the major drawbacks associated with the technique. Image reconstruction from projection data is usually computationally intensive because a large amount of projection data needs to be processed by time-consuming processing algorithms.

There are basically two types of CT image reconstruction methods, namely, analytic and iterative reconstruction algorithms [1]. An analytic reconstruction algorithm usually consists of two main computations. One is filtering and the other is backprojection. An iterative algorithm, however, starts with an initial guess of the solution (image) and iteratively updates (corrects) the image according to the measured projection data. No matter which variation of the iterative scheme is employed, the major computations in each iteration are forward (estimation) and backward (correction) projections which are equivalent to backprojection of an analytic scheme.

†This work was partially supported by NIH grant R01 CA51324.

*Also with Department of Electrical Science, Korea Advanced Institute of Science.

Various efforts have been made to shorten the reconstruction time, which can essentially fall into three categories, i.e., algorithmic improvement, dedicated hardware and parallel processing. In the first category, most of the attempts for iterative reconstruction methods have been concerned with reduction of the number of iterations, i.e., to make convergence faster. Examples include employing a larger correction factor in each updating phase [2-4], using coarser image (larger pixels) for the early iterations [5], and utilizing only a subset of the available projection data at each iteration [5], etc. In addition, FIR (filter-iteration reconstruction), an efficient method which extracts high-frequency components of the ratio of measured to computed projections and takes them into account for the iterative correction, was proposed [6]. On the other hand, fast backprojection algorithms have been developed for both of iterative and analytic reconstruction methods [7-8]. Two examples are the *STRETCH* algorithm [7], which approximates interpolation by using pre-computed look-up tables, and the *Incremental* algorithm [8], which restructures the conventional backprojection algorithm and performs backprojection on a beam-by-beam (instead of pixel-by-pixel) basis by using additions only except the first pixel in each beam.

In the second category, hardware techniques in different levels have been employed to speed up computations. For examples, a special hardware was used by Thompson et al. [9] to implement the approximated fast backprojection algorithm, *STRETCH*, mentioned above. Dedicated backprojectors were commercially designed to speed up backprojection [10]. By using microcoding techniques, Hart et al. [11] tried to do backprojection on an event-by-event basis to meet their critical time requirement (2 μs per event) in positron CT.

One of the major problems associated with the two approaches, algorithmic improvement and dedicated hardware, is that the achievable speed-up is quite limited while a tremendous amount of computation will be required for future 3-D CT systems. Therefore, it is believed to be imperative to rely on parallel processing techniques which have a potential to speed up reconstruction by several orders of magnitude. In other words, without employing multiple processing elements (PE's), the computational demand in 3-D CT would be formidable. However, only few studies on parallel processing approaches to CT image reconstruction have been attempted.

A parallel processing scheme was proposed by Jones et al.[12] for the EM (Expectation Maximization) algorithm. This scheme employed multiple PE's for forward and backward

0018-9499/90/0600-1333$01.00 © 1990 IEEE

projections which are major computations in iterative algorithms. Pixels are circulated through PE's, each of which backprojects one view of projection data by updating the appropriate pixels. The forward projection is performed in a similar fashion. However, they do not mention details of the parallel architecture and its control. A possible shortcoming of the scheme is low efficiency. For each PE involved in the back-projection, most of the pixels circulating through it are unnecessary. The same argument may be applied to the forward projection.

Another study by Llacer et al. [13] suggested a multiprocessor system for the MLE (Maximum Likelihood Estimator) algorithm. The system matrix, forward and backward projection vectors, etc., are partitioned into disjoint segments which are stored in different sections of memory. Each PE has a direct connection (access) to one of the sections. The connection is rotated through a crossbar switch such that each PE can have access to every section. In each step of the rotation, all PE's accumulate their partial forward and backward projections. However, crossbar switching is not desirable for a system with a large number of PE's. The number of PE's that may be employed is somewhat limited.

In this study, in order to check the feasibility of designing an efficient parallel processing system for future 3-D CT, we have implemented 3-D CT image reconstruction in parallel on the hypercube multiprocessor, iPSC/2, which is commercially available. The iPSC/2 employed consists of one controller and 32 PE's interconnected in the hypercube topology. In our implementation, only the analytic type of reconstruction algorithm, i.e., filtered backprojection, is considered. The overall reconstruction is decomposed into filtering and backprojection for which a two stage pipelining is adopted for each projection data set. In each stage, multiple PE's are employed for multiprocessing.

The filter kernel of 3-D CT system is symmetric with respect to both of horizontal and vertical axes in the projection domain. A conventional (sequential) convolution algorithm has been modified to exploit the symmetry of the filter kernel. That is, each product term (a coefficient * a projection data) is fully utilized once computed. This modified convolution is parallelized such that each PE deals with every Nth projection data when N PE's are employed in the convolution stage. Each projection data set is first sent from the controller to PE 0. PE 0, then, broadcasts this projection data set to N-1 other PE's. At the end of convolution of each projection data set, the filtered-projection data distributed among N PE's are integrated to a designated PE. Broadcasting and integration can be efficiently (in $O(\log N)$ steps) realized by using the (pseudo) binary tree structure embedded in the hypercube of N nodes (PE's) [14].

In the backprojection stage, the 3-D Incremental backprojection algorithm [8], which was recently developed and shown to be faster than the conventional algorithm, is parallelized. Backprojection is performed on a beam-by-beam basis in this scheme. Due to the limited number of PE's to be assigned to the backprojection stage, only coarse-grain parallelism is exploited. That is, each filtered projection domain (2-D) is partitioned into the same number of subdomains as that of PE's employed in this stage. Then, each PE backprojects a subdomain of filtered projection data into a 3-D array of image stored in its local memory. Thus, each PE in this stage will eventually produce a partially reconstructed 3-D image. Those partial (or local) results are integrated to the designated PE which sends the final reconstructed image to the controller.

The numbers of PE's in convolution and backprojection are determined by the processing time ratio of sequential convolution and backprojection algorithms such that a well-balanced load distribution is achieved between the two stages. In this implementation, the ratio varies from 6:1 to 20:1, depending on the image size. Due to the limited size of memory in each PE, our parallel reconstruction scheme has been tested only for image sizes of 31, 63, 95.

The remainder of this paper is organized as follows. The system geometry adopted in this study and the sequential 3-D CT reconstruction algorithm employed for parallelization are described in Section 2. The task partitioning schemes and communication patterns developed to optimize the efficiency of parallelization are discussed in Section 3. The implementation results and performance analysis are provided in Section 4. Then, conclusions of this study are given in Section 5.

2. Modification of Conventional Filtering and Backprojection

The major computational procedures in analytic reconstruction algorithms are filtering and backprojection. The projection data need to be filtered by the reconstruction filter before or after backprojection, which can be done either in the frequency domain or in the spatial domain. Filtering (using a filter obtained) in the frequency domain usually suffers from the *dishing* effect and a DC shift. In general, the frequency domain filtering is faster than the spatial domain convolution method because the FFT technique can be utilized. However, the former (frequency domain processing) would require more memory to implement due to zero padding and the constraint on the size of data (i.e., a power of 2). Even though the side effects in the frequency domain filtering can be minimized within an acceptable margin by modifying the filter, considering the memory capacity of the system to be used (Intel iPSC/2), we decided to perform 2-D convolution in the spatial domain. Therefore, the convolution backprojection scheme is adopted in this study, of which computational procedure can be represented as in Fig. 2.1. Note that the order of the functions may be exchanged.

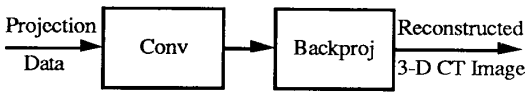

Fig. 2.1. Convolution backprojection scheme

A number of reconstruction algorithms have been proposed for the analytic reconstruction method. However, only few of them are concerned with practical 3-D CT systems. In [15], various types of reconstruction algorithms have been reviewed briefly. Of these, we adopt the concept of the Generalized True Three-Dimensional Reconstruction (GTTR) algorithm [16-18]. The reason is that it seems to be the only true 3-D algorithm considering the truncated practical 3-D CT system, which will be described in the following section. In the GTTR algorithm, the parallel beam reconstruction technique is employed rather than the cone beam reconstruction scheme. Therefore, through this paper, it is assumed that the projection data have been sorted (if necessary) into parallel beam sets.

In this section, we describe the conventional reconstruction algorithm for comparison with the modified reconstruction algorithms. A convolution algorithm is considered as a conventional one if it computes every product term expressed in the convolution formula. And those algorithms which perform backprojection on a voxel-by-voxel basis (or pixel-by-pixel for 2-D CT) are considered to be conventional backprojection algorithms, which may be represented by the *Shepp and Logan's algorithm* [19].

Conventional convolution and backprojection algorithms have been modified to minimize computational redundancy. The objective of modification is two-fold. One is to develop a faster sequential algorithm such that we can use a smaller number of processing elements (PE's) to achieve the same speed-up from parallelization, i.e., higher efficiency. The other is to have an algorithm which can be more efficiently parallelized.

2.1. System geometry

The 3-D CT system (scanner) which will be referred to in this study is a spherical system with its upper and lower parts truncated as illustrated in Fig. 2.2 The object under examination is to be placed along the Z-direction. With this geometry, the following terms are defined for concise and clear descriptions.

truncated ring : the rings at the upper and lower boundaries of the system sphere.
θ_m : the elevation angle of the truncated rings, which is the maximum θ for all rays,
view : a projection in a certain direction (ϕ,θ),
$R_{\phi,\theta}$: a parallel beam set in the direction (ϕ,θ),
$R_{\phi,(\bullet)}$: the set of $R_{\phi,\theta}$ sets for all θ and a certain ϕ,
$R_{(\bullet),\theta}$: the set of $R_{\phi,\theta}$ sets for all ϕ and a certain θ,
N_ϕ : the number of views in the azimuthal direction,
N_θ : the number of detectors between 0 and θ_m,
object : the interesting part of the body under examination, which is a sphere,
complete data set : a projection data set which covers a whole object,
incomplete data set : a projection data set which covers only a part of object.

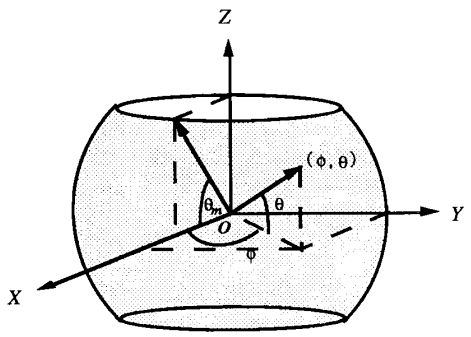

Fig. 2.2. System Geometry for 3-D CT system

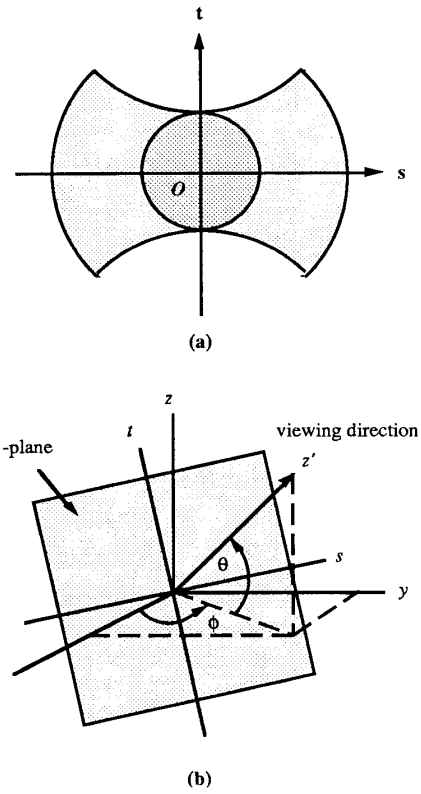

Fig. 2.3. (a) General shape of projection domain (b)*st*-plane and viewing direction by (ϕ, θ)

For a given size of object, it is easy to see that there exists an elevation angle θ_u such that $R_{\phi,\theta}$ is a complete data set if $\theta \le \theta_u$ and an incomplete data set otherwise. The general shape of a projection data domain is depicted in Fig. 2.3a. The (s, t, z') coordinate system is a rotated version of the (x, y, z) coordinate system as shown in Fig. 2.3b where the *st-plane* is perpendicular to the projection direction (ϕ, θ) and passes through the origin. For a complete data set, it is a full

circle, defined as *projection circle*. For an incomplete data set as shown in Fig. 2.3a, only the maximum inner circle in this projection data domain will be utilized in reconstruction. Although utilization of the incomplete data sets will eventually enhance the quality of the reconstructed 3-D CT images, it is beyond the scope of this study. For the purpose of feasibility examination, only complete projection data sets are under consideration.

2.2 Modified 2-D convolution algorithm

The conventional 2-D convolution algorithm is a well-defined technique, which is widely used in many different fields. Every projection data covered by the filter kernel will be multiplied by the corresponding coefficient to generate a product term as a contribution to the final convolved projection data. The design of a 2-D filter kernel has been one of the major difficulties in the 3-D CT image reconstruction for a truncated geometry. In this implementation, a filter kernel for the GTTR algorithm in the spatial domain will be used. The filter kernel is given as follows :

$$h_\theta(s,t) = S(s,t)\, h_p(s,t) \tag{2.1}$$

and

$$h_p(s,t) = \frac{-1}{(s^2 + t^2)^{3/2}} \tag{2.2}$$

where

$$S(s,t) = \begin{cases} 1 & \text{for } \left| \tan^{-1}\left(\frac{t}{s}\right) \right| \le \gamma_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

and

$$\gamma_{max} = \cos^{-1}\left(\frac{\cos\theta_u}{\cos\theta}\right) \tag{2.4}$$

Since the shape of a projection data set is a circle, the 2-D convolution is carried out within the projection circle instead of a square containing this circle. Although this incurs overhead of checking whether a projection data is inside the circle, computations for more than 22% of projection data can be saved. The conventional convolution algorithm is provided for reference, which basically implements the following equation as it is, where g(m,n) is the convolved projection data and p(i,j) is the projection data.

$$g(m,n) = \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} h_\theta(m-i,\, n-j)\, p(i,j), \quad -\frac{N}{2} \le m,\, n \le \frac{N}{2} \tag{2.5}$$

Modification of conventional 2-D convolution algorithm is based on the symmetry of the 2-D filter kernel employed in this study. From Eqs. (2.1) - (2.4), it is easy to see that the filter kernel $h_\theta(s,t)$ is symmetric with respect to *s*-axis and *t*-axis. That is, g(m,n), g(2i-m,n), g(m,2j-n) and g(2i-m,2j-n) include the same product term $h_\theta(m-i,n-j)p(i,j)$. For the convenience, let a *k-group* associated with the product term $h_\theta(m-i,n-j)p(i,j)$ be defined as the group constituted by these four convolved projection data. The key idea of the modified 2-D convolution algorithm is to utilize each product term as much as possible once it has been computed. In other words, if a product term has been computed for one convolved projection data, say g(m,n), it can be used to update the other three members in the associated k-group. Note that if i=m or j=n, there are only two members in a k-group. One major difficulty in implementing this idea is that we need to provide an efficient scheme to check whether a product term for a convolved projection data, say g(m,n), has already been computed. To solve this problem, the following two schemes are used to generate a regular load pattern for each g(m,n) as shown in Fig. 2.5.

The first scheme is to utilize the geometrical symmetry in the projection circle. Referring to Eq. (2.5), we consider g(m,n), g(n,m), g(-n,m), g(-m,n), g(-m,-n), g(-n,-m), g(n,-m) and g(m,-n) together as a group, where g(m,n) is assumed to be in the first octant of projection circle. For the convenience, let *g-group* associated with g(m,n) be defined as the group containing these eight convolved projection data. That is, if a product term, say $h_\theta(m-i,n-j)p(i,j)$, is computed for g(m,n), those corresponding product terms for the other members in the same g-group, e.g., $h_\theta(n-j,m-i)p(j,i)$ for g(n,m), are computed before any other product term for g(m,n) is computed. Note that if i=j, or i=0, or j=0, there are only four members in this group.

Since only those g(m,n)'s inside the projection circle are to be computed, there are basically two kinds of checking operations required for the modified 2-D convolution algorithm. The first checking rule, called *g-checking*, states that a product term $h_\theta(m-i,n-j)p(i,j)$ is computed only when both of g(m,n) and p(i,j) are inside the projection circle. The other checking rule, called *k-checking*, states that a member in a k-group is updated by the associated product term only when this member is inside the projection circle. It is clear that one of the advantages of utilizing geometrical symmetry is that those information derived from g-checking and k-checking can be shared by all the members in the same g-group.

The other scheme to generate a load pattern of regular shape is to perform the convolution from the center toward the outer part of the projection circle. Specifically, our approach is to partition the area in the projection circle into *circles* concentric with the projection circle. The radius difference of every two adjacent circles, say circle *l* and circle *l*+1, is one projection data interval. Those g(m,n)'s in the inner most concentric circle, circle 1, are processed first and then those g(m,n)'s between the circle 1 and circle 2 are processed next. In general, those g(m,n)'s between circle *l* and circle *l*+1 are computed right after those g(m,n)'s between circle *l*-1 and circle *l*.

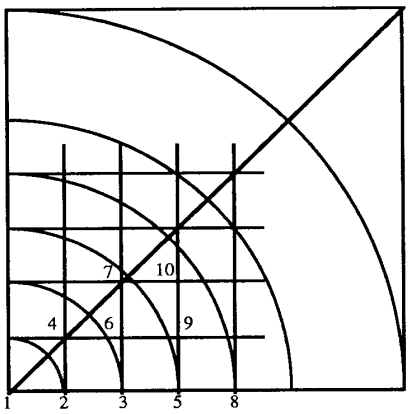

Fig. 2.4. Processing sequence in the Modified 2-D convolution algorithm indicated by the numbers beside the grids

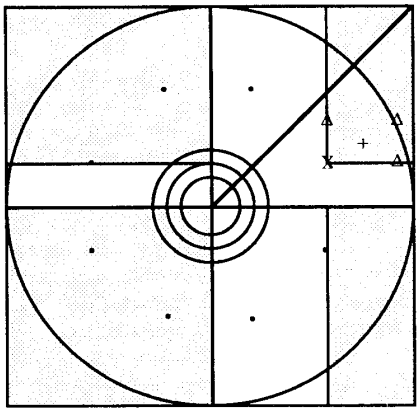

Fig. 2.5. Load pattern, *g-group* and *k-group* for a convolved projection data X in the first octant of projection circle

These two schemes are illustrated in Fig. 2.4 and 2.5. In Fig. 2.4, where only the first quarter of the projection circle is shown, the processing sequence in the first octant is depicted. The numbers beside grids indicate the processing sequence employed in the modified 2-D convolution algorithm. Recall that we only need to consider the processing sequence in the first octant because of the g-group scheme. In Fig. 2.5, the regular load pattern for convolved projection data X is indicated by the shaded region (excluding those areas outside the projection circle). The other seven members in the g-group associated with X are indicated by "•". The "+" represents a product term and "Δ" denotes the other three members, which need the product term, in the k-group associated with X and "+".

Based on these two schemes discussed above, the modified 2-D convolution algorithm, *Algorithm MC2*, is sketched as follows.

Algorithm MC2

Following the processing sequence do
  let center of filter kernel be located at (m,n)
  IF g(m,n) is inside the projection circle DO
  let h_θ(m-i,n-j)p(i,j) be the product term to be computed
    IF p(i,j) is inside the projection circle DO
    compute this product term and corresponding product terms for the members in the *g-group* associated with g(m,n)
    update g(m,n) and all other members in the same *g-group* using corresponding product terms
    ENDIF
    FOR all three other members in the *k-group* associated with g(m,n) DO
    consider one of these three member, say g(k,l)
      IF g(k,l) is inside the projection circle DO
      update g(k,l) and all other members in the same *g-group* associated with g(k,l) by the corresponding product terms.
      ENDIF
    END FOR
  ENDIF
END

It is easy to see that one of the potential disadvantages of this algorithm is that more g(m,n)'s are required to be checked whether they are inside the projection circle than a conventional convolution algorithm due to k-checking. However, due to full utilization of every product term once it is computed, the number of multiplications can be reduced substantially. The actual benefit obtained from reducing the number of multiplications depends on the ratio of multiplication time to addition time in a particular machine. A performance analysis can be found in [20].

2.3 Incremental Backprojection Algorithm

Backprojection is an essential procedure in CT image reconstruction, which is to deal with a huge amount of projection data. In a conventional algorithm, which may be represented by the Shepp and Logan's algorithm, $O(N)$ sine (cosine) computations, and $O(N^2)$ multiplications and additions, are required for each view to be backprojected onto a 2-D CT image of size N * N. For 3-D CT, the complexities of these computations further increase to $O(N^2)$ for sine (cosine) computations and $O(N^3)$ for multiplications and additions. Therefore, in order to have a fast CT system, it is essential to reduce the backprojection time.

As an attempt to reduce the number of sine (cosine) computations and multiplications, a new fast backprojection

scheme, *Incremental algorithm* [8], to be employed in this parallel implementation, has been developed by restructuring the conventional backprojection algorithms which perform backprojection on a pixel-by-pixel basis. To minimize the computational redundancy, the inter-dependency of pixel computations (*position* and *value*) is transformed to a set of incremental relations for each beam where a beam is a set of pixels enclosed by two adjacent rays in 2-D CT and a set of voxels enclosed by four adjacent rays in 3-D CT. This Incremental backprojection algorithm should be distinguished from the incremental ray tracing algorithms in computer graphics [22] in that it is not only to locate pixels within each beam but also to compute pixel values. The idea is that once the pixel value and position of a (first) pixel in a beam are computed, those of all the other pixels in the same beam can be derived successively by *additions* only with some *beam constants*. The beam constants for a beam are determined by the values of the two rays enclosing the beam and the positional relation of two adjacent pixels in the beam. Notice that there can be only few different positional relations. Therefore, these beam constants can be computed in advance for each beam and applied to all the pixels (except the first) in the beam. As a result, backprojection is naturally performed on a beam-by-beam basis. The complete description and performance analysis of this backprojection scheme can be found in [8].

3. Parallelization

Parallel implementation of 3-D CT image reconstruction has been realized on a hypercube multiprocessor system, Intel iPSC/2. The newly developed algorithms, the modified 2-D convolution and 3-D Incremental algorithm, have been adopted to maximize speed-up for a given number of processing elements (PE's). As mentioned earlier, the main concern of this study is to examine the feasibility of parallelization of the 3-D CT image reconstruction. Therefore, parallelism involved in the reconstruction algorithms has been utilized as much as possible to provide sufficient information for future study on designing a dedicated parallel system. In this study, we aim at not only a large speed-up but also a reasonable efficiency, where *efficiency* is defined as the ratio of a speed-up to the number of processing elements. Also, the effect of load balancing on the overall performance has been studied.

There are basically three types of potential parallelism involved in the 3-D CT reconstruction algorithm, which can be exploited by a coarse grain parallel architecture like iPSC/2. Hierarchically, they are view, data, and function parallelism. The view parallelism, which is on the highest level, allows independent processing of different views. That is, since no interactions are required among different views during reconstruction, all views can be processed simultaneously in theory. However, using $N_\phi * N_\Theta$ PE's is usually impractical in a parallel implementation, especially for a coarse grained system. Furthermore, the size of an incomplete projection data set varies with θ. If they are processed at the same time, the load balancing could be a potential problem. Therefore, the view parallelism is to be utilized in such a way that several $R_{\phi,(\bullet)}$'s are assigned to a set of PE's and, among each set of PE's, those views with the same θ but different ϕ are processed simultaneously. In fact, due to the limited number (32) of PE's available in the iPSC/2, the view parallelism is not exploited in this study. Instead, only one set which contains all the available PE's is employed to get a partially reconstructed image from a $R_{\phi,(\bullet)}$ in this study. If more PE's (sets of PE's) are available, this scheme can easily be replicated to process all other $R_{\phi,(\bullet)}$'s and the partially reconstructed images from the sets may be efficiently combined through a fast communication channel.

The second level parallelism, the data parallelism, is mainly concerned with the independency among different projection data in a view. In the modified 2-D convolution algorithm, it is not necessary for a g(m,n) in the processing sequence to be processed after other g(m,n)'s for they are independent with one another. Similarly, all beams can be backprojected at the same time if necessary. Therefore, it is reasonable to process a number of g(m,n)'s and/or beams at the same time.

The other parallelism is function parallelism. Execution of the two functions, convolution and backprojection, can be overlapped. That is, while backprojection being performed for one view, convolution for another view may be carried out at the same time. In order to exploit the function parallelism, a *two stage pipelining* approach is employed in this study, in which a convolution stage is followed by a backprojection stage. Since we need to process a large number of projection data sets, this two stage pipeline can be utilized efficiently.

Within each stage, the data parallelism is exploited by multiple PE's. The number of PE's for each stage is determined as follows. We first compute the ratio, γ, of sequential convolution time to sequential backprojection time. Suppose that $k < \gamma < k+1$, where k is an integer. Then, k PE's are assigned to the convolution stage and one PE to backprojection stage. In addition to (k, 1), the combination of processing elements between stages, we also consider another two combinations (k-1, 1) and (k+1, 1) to study the effect of the overhead due to communication among PE's. More generally, we examine combinations (n_c, n_b) such that $n_c + n_b \leq 32$ and $n_c \in \{k', k'+1, k'-1\}$ where n_c denotes the number of convolution PE's, n_b denotes the number of backprojection PE's and k' is an integer satisfying $k' < n_b \times \gamma < k'+1$. Note that the ratio γ varies with the size of an image. As in the sequential implementation, the sizes of images under test are 31, 63, and 95.

In the remainder of this section, task partitioning is discussed in detail for each of convolution and backprojection. Then, efficient communication schemes which can realize data exchange required among subtasks are described. Finally, extensive implementation results are provided with detailed discussions.

3.1. Task partitioning in the convolution stage

For a conventional 2-D convolution algorithm which usually deals with a square domain of data set, the data domain can easily be partitioned into blocks or strips with well balanced load distribution. However, since the projection data domain is a circle in our case (refer to Section 2.1), a horizontal or vertical decomposition would not be an appropriate approach. A reasonable solution is to partition the data domain according to the convolution processing sequence. The processing sequence, as illustrated in Fig. 2.4, is a sequence to perform convolution from the center to the boundary of the projection circle. Recall that only g(m,n) (convolved projection data) in the first octant of projection circle are in the sequence and all the members of a g-group associated with a g(m,n) are processed together with this g(m,n). That is, conceptually, the processing sequence is a sequence for the g-groups, not only for a g(m,n) in the first octant but also for all members in the corresponding g-group.

Suppose there are n_c PE's in the convolution stage and the processing sequence is $(p_1, p_2, ..., p_t)$, where p_j is the jth g(m,n) in the sequence and t is the total number of g(m,n) in the first octant of the projection circle. Then PE_i is assigned to perform convolution for the following set of g(m,n), which is denoted by {P(i)}.

$$\{P(i)\} = \{ p_j \mid j = n_c \times k + i, k = 0, 1, ...\}$$

In other words, the PE_i is responsible for performing convolution for every n_c-th g-group in the processing sequence. In addition to simplicity, an obvious advantage of this approach is that it is easily applicable for different sizes of images and different numbers of PE's employed.

Even if each PE only deals with a part of the convolution task for a view, it still requires a whole projection data set and a complete filter kernel (Recall that the filter kernel is different for different θ, referred to Eqs. (2.1) - (2.4).). Due to the limited memory size in each node of iPSC/2, we can not store all sets of the projection data and the filter kernel in the local memory of each PE. To solve this problem, these data are sent to convolution PE's view by view, i.e., the cubemanager of iPSC/2 is in charge of sending a projection data set and filter kernel to the convolution PE's for the convolution of each view. It is easy to see that, in order to minimize the interactions among PE's during computation, each PE must have a copy of filter kernel and a projection data set, which might be an disadvantage if memory size is a major concern.

The above simple partitioning scheme cannot guarantee a uniform load distribution among PE's. Remember that the computational load of g(m,n) varies with m and n. However, it is reasonable to expect an acceptable load distribution due to the following reasons.

In general, g(m,n) in the inner part of the projection circle have much more load than g(m,n) in the outer part. It can be easily checked by the size of load pattern. Furthermore, the size (area) of load pattern decreases gradually from center (except the origin) to the boundary. It implies that the load distribution corresponding to the processing sequence is roughly monotonically decreasing with the exception that g(m,m) and g(m,0) have a smaller load. The exception is due to the fact that there are only four members in the g-groups associated with those convolved projection data on the s-axis and when m=n. Therefore, when the size of image is large, we may expect that the computational load is approximately uniform for all {P(i)}.

3.2. Task partitioning in the backprojection stage

Since the ratio γ of the sequential convolution time to the sequential backprojection time is quite large and there are only 32 PE's in iPSC/2, at most four PE's may be employed for backprojection. We have considered three cases where one, two or four PE's is (are) assigned to the backprojection stage. When there are two PE's employed in the backprojection stage, the object is divided into four spaces by the sz'-plane and tz'-plane (refer to Fig. 2.3b). These four spaces are grouped into two volumes. One volume is composed of the two spaces containing those beams formed by the convolved projection data in the first (positive s and t) and the third quadrants (negative s and t) of the projection circle. The other volume consists of the other two spaces. Each PE in the backprojection stage processes one of the two volumes. Note that pair-by-pair backprojection is adopted in the 3-D Incremental algorithm, so those voxels symmetric with respect to the origin will be backprojected together [8].

If four PE's are employed, the object is further divided by the st-plane. In order to clearly describe the eight spaces in the target sphere, we will use the convention that if a space contains voxels with positive s, negative t and positive z', it is denoted by space($+s$, $-t$, $+z'$). Following this convention, the backprojection space is grouped into four volumes as follows ;

- space($+s$, $+t$, $+z'$) and space($-s$, $-t$, $-z'$),
- space($+s$, $+t$, $-z'$) and space($-s$, $-t$, $+z'$),
- space($+s$, $-t$, $+z'$) and space($-s$, $+t$, $-z'$),
- space($+s$, $-t$, $-z'$) and space($-s$, $+t$, $+z'$).

Each volume is assigned to a PE in the backprojection stage. Note that each space represents one eighth of the whole backprojection load.

Obviously, both task partitioning, i.e., when there are two and four PE's, can result in a balanced load distribution since each PE is assigned the same size of volume. It is worth pointing out that the reason why it is possible to further divide the space along the st-plane is the 3-D searching flow scheme developed for the 3-D Incremental algorithm [8], which performs searching from the st-plane (center of beams) forward and backward toward the boundary of the object

respectively. On the other hand, however, since the beam length is reduced to one half of the original length, the beam constants are not utilized so efficiently as before. Therefore, the performance of the 3-D Incremental algorithm would be degraded [8].

A potential disadvantage of this approach is that since each PE performs backprojection on a beam-by-beam basis, it requires voxels in different part of the target sphere. If each PE has only a subimage in its local memory, all PE's in the backprojection stage have to exchange some part of the subimages in their own local memories for every view. It will introduce a new overhead. A reasonable alternative is to allow each PE to maintain a complete partially reconstructed target sphere and sum them up after the reconstructions for all views are finished. However, a large size of memory required in this scheme is a potential disadvantage if the memory capacity is limited.

3.3. Communication patterns

There are basically two different data flows involved in this implementation, namely, *broadcasting* and *integration*. A projection data set and a filter kernel associated with a particular θ are first sent from the cubemanager of iPSC/2 to PE_0 in the convolution stage. PE_0 then broadcasts the projection data set and filter kernel to the other convolution PE's. After the convolution for a view is finished, the convolved projection data distributed among all convolution PE's are integrated into one of the PE's, called *gate node*, in the backprojection stage. From the gate node, the convolved projection data is broadcast to the other backprojection PE's. The partially reconstructed target spheres are integrated into a PE, called *return node*, in the backprojection stage after the reconstruction for all views is completed. And the final reconstructed target image is sent back to the cubemanager from the return node.

Since the number of PE's which can be allocated in iPSC/2 for every application must be a power of 2, we have employed only three different sizes of cubes, 8, 16, and 32 in this study. For a cube of size k, the PE's are numbered from 0 to k-1. In general, the first n_c PE's are employed for convolution and the last n_b PE's for backprojection if the combination (number of convolution PE's, number of backprojection PE's) under test is (n_c, n_b). Also, PE_{k-1} is chosen to be the gate node and PE_{k-n_c} to be the return node.

The main concern in realizing the communication pattern is how to complete a data flow (broadcasting or integration) in as few steps as possible. The iPSC/2 supports the circuit switching data communication. A physical path established between any two PE's before the actual data transfer is maintained until the communication is completed. Therefore, if there is a common link between any two communication paths, one of the communication must wait until the other is finished, where a link is the physical connection between two adjacent PE's. In order to minimize such conflicts among communication paths, a localized communication should be employed as much as possible in designing a communication pattern.

A binary tree structure is known to be one of the most efficient topologies for integrating local results. In general, it takes $\log_2 N$ steps for N items. Of course, a common bus structure could provide the fastest broadcasting. However, since there is no common bus supported by the iPSC/2, a reasonable alternative for broadcasting is also to use binary tree structure. Two different approaches to implementing a binary tree on a hypercube topology may be considered. One is to embed a (complete) binary tree into a hypercube directly [21]. In this case, the utilization of PE's in the hypercube is quite low (below 50%). Although it has the advantage that each PE is required to be involved in only three communications (two with its sons and one with its parent) so that it can become available for other tasks immediately, this approach is not suitable for our study. It is because there is a limited number of PE's in iPSC/2 (32 PE's) and in most cases we need more than 16 PE's which can not be offered by this scheme due to low utilization.

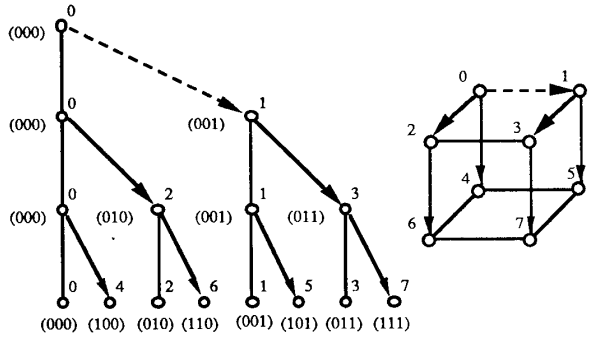

Fig. 3.1. A 4-level pseudo binary embedded in a 3-cube

The other approach is to embed a pseudo binary tree into a hypercube [14]. In this scheme, a binary tree structure is embedded in such a way that each PE might represent more than one node in the binary tree. As an example, a 4-level binary tree embedded in a binary 3-cube for broadcasting, is illustrated in Fig. 3.1 where, e.g., the PE_0 represents four nodes in the binary tree. Because of its high utilization of PE's in the hypercube, we adopted pseudo binary trees. Although it does have the problem that some PE's like PE_0 in Fig. 3.1 need to be involved in communication for a long time, it can be partially alleviated by overlapping broadcasting and integration as will be discussed later. Note that every communication is carried out along a single link, by which conflicts among different communication paths can be avoided.

There are two different data flows, broadcasting and integration, which are executed one after another. In order to minimize communication overhead, we overlap these two data flows. The idea is to make the difference d of node numbers

between PE pairs (source-destination), called *difference*, decrease by the factor of 2, i.e, d to $d/2$, for each step of broadcasting and increase by the factor of 2, i.e., d to $2d$, for each step of integration. The data (e.g., local results) are integrated into odd PE's (PE's with odd PE numbers) in the first step. The advantage of this design is that after the first step of an integration, the even PE's (PE's with even PE numbers) can be involved in broadcasting for next view. In Fig. 3.2, the broadcasting pattern implemented on a pseudo binary tree for 13 convolution PE's is shown, in which the difference decreases from 8 to 1. In Fig. 3.3 the integration pattern for 13 convolution PE's and the gate node (PE_{15}) is shown, in which the difference increases from 1 to 8. Note that from the second step in the integration, only odd PE's are involved and they are not required for broadcasting until the last step.

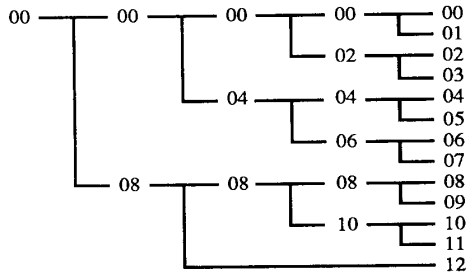

Fig. 3.2. Broadcasting pattern for 13 convolution PE's

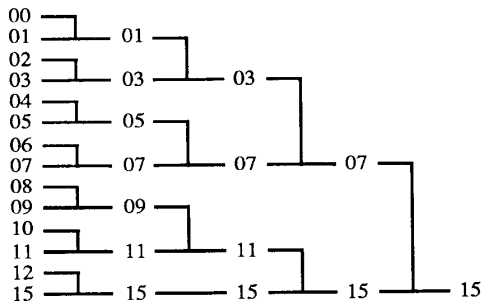

Fig. 3.3. Integration pattern for 13 convolution PE's and the gate node (PE_{15})

An inherent problem in the integration pattern is that sometimes the destination PE of a PE pair does not exist in some steps. To cope with this problem, we pair up the source PE directly with the gate node. For example, PE_{13} is not involved , so PE_{12} is paired up with the gate node in Fig. 3.3. Although the *abnormal communication path* for this new pair is no longer localized, the communication paths are still exclusive with each other.

For broadcasting and integration in the backprojection stage, we employ the same idea as in the convolution stage. However, since the number of backprojection PE's is always a power of 2, we do not have the abnormal communication path in this stage.

In the following, the general algorithms for broadcasting and integration using a pseudo binary tree are provided where $\oplus$ denotes binary exclusive OR.

Broadcasting

```
/* suppose the number of this PE is k */
l = number of levels in the pseudo binary tree - 2
let PE[k] denote the PE_k
WHILE (l ≥ 0)
    IF k < k ⊕ 2^l DO
        send data to PE[k ⊕ 2^l]
    ELSE
        receive data from PE[k ⊕ 2^l]
    ENDIF
    l = l - 1
END WHILE
```

Integration

```
/* suppose the number of this PE is k */
L = number of levels in the binary tree - 2
let PE[k] denote the PE_k
l = 0
WHILE (l ≤ L)
    IF k < k ⊕ 2^l DO
        if k ⊕ 2^l < n_c send data to PE[k ⊕ 2^l]
        else send data to gate node ( PE[2^(L+1) -1])
    ELSE
        receive data from PE[k ⊕ 2^l]
    ENDIF
    l = l + 1
END WHILE
```

4. Implementation results and discussions

To examine the effect of image size on the performance of parallel processing, we have considered three different sizes of images, 31, 63 and 95. For each of them, different combinations of the number of convolution PE's and that of backprojection PE's (n_c, n_b) have been tried. The number of PE's assigned to each stage is determined by the ratio γ where γ is the ratio of sequential convolution time to sequential backprojection time as defined in Section 3.1. The possible combinations which can be realized in iPSC/2 (with 32 PE's) for different sizes of images are listed below.

image size 31 : (5, 1), (6, 1), (7, 1), (12, 2), (13, 2), (14, 2), (26, 4), (27, 4), (28, 4)
image size 63 : (14, 1), (15, 1), (16, 1), (28, 2), (29, 2), (30, 2)
image size 95 : (23, 1), (24, 1), (25, 1)

Table 4.1. Overall implementation results before load balancing

N	n_c/n_b	Tn	Sp	Ef	Um	Us
31	5/1	6	5.13	85.51%	98.77%	0.85%
31	6/1	7	6.16	87.95%	98.41%	0.93%
31	7/1	8	6.76	84.46%	98.33%	5.29%
31	12/2	14	11.40	81.45%	92.83%	5.54%
31	13/2	15	11.53	76.87%	89.80%	7.93%
31	14/2	16	12.21	76.29%	89.55%	8.14%
31	26/4	30	18.94	63.12%	82.29%	12.35%
31	27/4	31	19.66	63.42%	82.85%	11.03%
31	28/4	32	19.33	60.39%	79.53%	12.86%
63	14/1	15	13.33	88.86%	96.84%	2.61%
63	15/1	16	14.26	89.09%	97.30%	1.93%
63	16/1	17	15.25	89.73%	96.10%	3.84%
63	28/2	30	24.29	80.97%	92.56%	4.60%
63	29/2	31	25.02	80.81%	92.55%	5.68%
63	30/2	32	26.03	81.35%	92.69%	4.35%
95	23/1	24	21.20	88.35%	96.49%	2.37%
95	24/1	25	22.39	89.56%	96.56%	2.55%
95	25/1	26	23.06	88.70%	97.56%	2.36%

In Table 4.1., the implementation results for these 18 different cases are provided, where

- N : image size, i.e., size of $N \times N \times N$,
- n_c : number of convolution PE's,
- n_b : number of backprojection PE's,
- Tn : $n_c + n_b$,
- Sp : speed-up, defined as the ratio of sequential processing time to parallel processing time,
- Ef : efficiency, defined as Sp/Tn,
- Um : mean value of utilization for all convolution PE's, where utilization for a convolution PE is defined as the ratio of the net computation time to the total processing time of the PE,
- Us : standard deviation of utilization for all convolution PE's.

In general, the Um reflects the average usage of all convolution PE's devoted to the pure computation and Us indicates how uniformly the PE's are utilized for the convolution. If Us is large, it means that the computational load for convolution is not well distributed among PE's. Note that a lower utilization of a PE does not necessarily imply a smaller computational load for the PE than those for other PE's. An extreme example is that every PE does have the same computational load but different communication load which includes broadcasting, integration, and delay due to improper data flow in our study. On the other hand, efficiency Ef represents the overall percentage of PE utilization dedicated to the computational load, including convolution and backprojection.

From Table 4.1., it can be observed, first of all, that the 3-D CT image reconstruction can be efficiently parallelized on a hypercube. High speed-up and efficiency were achieved in most cases considered. For example, the speed-up for the image size of 95 with the combination (25, 1) is as high as 23.06 with a promising efficiency of 88.70%. In general, it may be observed that the larger the image size is, the larger speed up with a higher efficiency can be achieved for a given number of PE's. Also, the more PE's are employed, the lower efficiency results for a given image size, though the speed-up might be higher. For example, for the same total number of PE's, say 30, 31 or 32, a larger speed-up is obtained for the image size of 63 than for the image size of 31. And for image size of 31, when the number of PE's is less than 10, efficiency is above 80%, and when the number of PE's is larger than 30, efficiency is as low as 60%.

Although the results are quite appealing in most cases, several clues have suggested that this parallelization still has rooms for improvement. The obvious one is that when the Us increases, the efficiency generally becomes lower for a certain image size. However, it is not clear if low PE utilization (some PE utilization must be relatively low, to have a large standard deviation) comes from unbalanced computational load or communication load. Another clue is that, in the case of image size of 31, speed-up is even smaller when Tn is 32 than when Tn is 31. It means that the overhead due to either communication or unbalanced load distribution becomes more influential than the potential speed-up obtainable by employing more PE's. The third clue is that Um is much larger than Ef. Since efficiency, Ef, represents the average PE utilization for both stages, it means that we may have either quite an unbalanced distribution of PE utilization or significant broadcasting time in backprojection stage, or the processing times of two stages are quite different. For detailed discussions, we provide the load distribution among PE's for one of the 18 different cases in Table 4.2. The load distributions for the other 17 different cases can be found in Appendix A of [20].

In these tables, the first column is the PE number. The second column, *1st brdcst*, indicates the time for each PE to acquire the first data set after a synchronization signal used to synchronize all PE's at starting point. The third column, *brdcst*, gives the total broadcasting time except the first data set. The fourth column, *integration*, provides the total integration time. The fifth column, I/O, shows the amount of time PE_0 spent in receiving the first data set from the cubemanager plus that the return node used to send the completely reconstructed target sphere back to the cubemanager. Then the sixth column, *compute*, gives the net

time devoted to the computational load. And the seventh column, *node time*, shows the total processing time for each PE and the last column is the ratio of computation time to node time which indicates the PE utilization for the computation.

Table 4.2. Load distribution for N=31 and (n_c , n_b) = (28, 4) before load balancing

node	1st brdcst	brdcst	integrate	I/O	compute	node time	utilize
0	104	711	43	33	6718	7257	92.57%
1	104	26	329	0	7840	8303	94.42%
2	104	302	41	0	8647	9098	95.04%
3	104	26	1449	0	7542	9125	82.65%
4	104	305	42	0	7231	7686	94.08%
5	104	25	329	0	9676	10138	95.44%
6	104	349	42	0	8595	9096	94.49%
7	104	26	2056	0	7975	10165	78.46%
8	104	300	40	0	9510	9959	95.49%
9	104	29	1173	0	8664	9973	86.87%
10	104	2767	43	0	6925	9841	70.37%
11	104	25	1424	0	8430	9987	84.41%
12	104	1626	44	0	8118	9895	82.04%
13	104	561	329	0	8951	9948	89.98%
14	104	2281	44	0	7431	9864	75.33%
15	104	27	2952	0	7092	10178	69.68%
16	104	304	40	0	8528	8981	94.96%
17	104	25	982	0	7881	8995	87.62%
18	104	2054	43	0	6810	9015	75.54%
19	104	845	2436	0	5654	9043	62.52%
20	104	3861	41	0	4801	8812	54.48%
21	104	2818	331	0	5607	8862	63.27%
22	104	2300	42	0	6555	9004	72.80%
23	104	539	1311	0	7109	9068	78.40%
24	101	2797	41	0	6893	9837	70.07%
25	101	2496	1870	0	5380	9852	54.61%
26	101	3052	40	0	6626	9825	67.44%
27	101	2199	1279	0	6282	9865	63.68%
31	614	1832	43	0	8120	10613	76.51%
30	614	1287	43	0	8691	10638	81.70%
29	614	1871	596	0	8080	11165	72.37%
28	410	1349	1091	45	8438	11337	74.43%

Table 4.2., which shows the load distribution for the image size of 31 and combination (28, 4), provides an example that the computational load is not well balanced. For instance, computational load for some PE is as high as 9676 but for another PE is so low as 4801. It means that unbalanced load distribution is one of the major reasons for relatively low efficiency in some cases. Another phenomenon that can be observed is that the load distribution in the backprojection stage is relatively balanced. However, since the total processing time required in the convolution stage is determined by the PE with the heaviest computational load which may become larger due to the unbalanced computational load distribution, the total backprojection time has been increased because of waiting for delayed convolved projection data. This results in low PE utilization in backprojection stage and in turn low efficiency for overall system performance.

Based on the above discussions, it may be said that the simple task partitioning scheme for the convolution stage might produce an unbalanced computational load distribution when the image size is small and too many PE's are employed. Note that it works well for a large image size even if we use more than 20 PE's. In order to investigate how computational load balancing affects the overall performance, we attempted to improve the task partitioning scheme for the convolution stage as described in the following.

Fine tuning of computational load balance

Although we expect to have a reasonable load distribution from partitioning the processing sequence of convolution which is roughly sorted in nature, it is still possible that some PE happens to do convolution for more pixels on the s-axis or the line $s=t$ than the others. It will lead to a less computational load for the PE. To avoid this possibility, we sort the processing sequence in the order of nonincreasing computational load associated with those g-groups in the first octant of the projection circle. And then, in order to get a near-optimum load distribution, we assign load to each PE in the following way.

If there are n_c PE's in the convolution stage, n_c load-queues are created such that each of them keeps a list of g-groups to be processed by a PE. Initially, all queues are empty. In each step, we assign the g-group with the heaviest load in the sorted processing sequence to the queue with the least total load and delete this g-group from the sorted processing sequence.

Since the objective is to examine how computational load balancing affects the system performance, we do not include the overhead introduced by this new task partitioning scheme for the convolution stage in the following discussions. Note that this is a static load balancing which can be done in advance. We have remeasured the performance for the same 18 different cases. The overall implementation result is shown in Table 4.3. and the load distributions for two of the 18 different cases are provided in Tables 4.4. - 4.5. The load distributions of the other 16 different cases can be found in Appendix B of [20].

Apparently, all performance figures Sp, Ef, Um, and Us have been improved. In other words, we have successfully achieved a better computational load distribution than before by this task partitioning scheme. For example, in the case of the image size of 31 and combination (28, 4), (refer to Table. 4.5.), the computational load is more balanced than before and also the communication time has become quite small in general. In Fig. 4.1, we show the improvements of Um and Us due to load balancing, which are denoted by delta Um and delta Us. The horizontal axis in this figure indicates the 18 different cases in the same order as in Tables 4.1. and 4.3. It is clear that, generally, the Um and Us exhibit a similar tendency in improvement. And for some cases like the image size of 31 with combinations (26, 4), (27, 4) and (28, 4) where efficiency was quite low before load balancing, the improvements are quite significant.

Table 4.3. Overall implementation results after load balancing

N	n_c/n_b	Tn	Sp	Ef	Um	Us
31	5/1	6	5.38	89.59%	98.83%	0.37%
31	6/1	7	6.42	91.68%	98.34%	0.32%
31	7/1	8	7.26	90.79%	98.15%	0.69%
31	12/2	14	11.57	82.61%	96.32%	1.65%
31	13/2	15	12.52	83.45%	95.04%	2.11%
31	14/2	16	13.10	81.89%	93.33%	1.60%
31	26/4	30	20.59	68.63%	92.73%	2.25%
31	27/4	31	20.85	67.27%	91.80%	2.80%
31	28/4	32	21.15	66.08%	90.43%	3.42%
63	14/1	15	13.85	92.35%	98.57%	0.56%
63	15/1	16	14.78	92.35%	98.41%	0.62%
63	16/1	17	15.53	91.33%	98.48%	0.57%
63	28/2	30	25.57	85.24%	96.35%	0.99%
63	29/2	31	26.60	85.80%	96.92%	1.21%
63	30/2	32	27.04	84.49%	96.02%	1.40%
95	23/1	24	22.28	92.82%	98.51%	0.34%
95	24/1	25	23.06	92.24%	98.86%	0.38%
95	25/1	26	23.96	92.16%	98.95%	0.31%

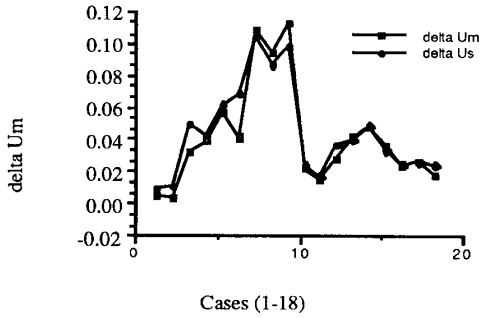

Fig. 4.1. Improvements on Um and Us.

Fig. 4.2. Improvements on Ef and Us.

Also, in Fig. 4.2, the improvements of Us and Ef are plotted for comparison. However, a substantial improvement of Us does not imply a significant improvement in Ef in these cases. It appears that the possible improvement is limited to be around 6% at most. In addition, from Table 4.3., we can see that the convolution PE's are almost devoted to the computational load, but efficiency is still much lower than the Um for some cases.

Table 4.4. Load distribution for N=31 and (n_c , n_b) = (5, 1) after load balancing

node	1st brdcst	brdcst	integrate	I/O	compute	node time	utilize
0	100	619	42	32	39417	39861	98.89%
1	99	25	361	0	39391	39882	98.77%
2	100	94	42	0	39917	40158	99.40%
3	100	25	587	0	40049	40764	98.25%
4	93	317	44	0	39395	39854	98.85%
7	1907	11117	0	45	29044	42119	68.96%

To explain this observation, we found out that we need to not only evenly distribute computational load but also arrange the load such that the heavier load is assigned as close to the later steps as possible in integration. It is because the waiting time for receiving data is accumulated during integrations. For instance, in Table 4.5., the computational load for PE_6 is relatively high among PE's, but unfortunately it is a source PE in the first step. So, it becomes a bottleneck, which forces PE_7 to wait for PE_6 finishing, though PE_7 is much faster. We can see that PE_7 has relatively high integration time. Note that the waiting time of PE_7 is not only the difference of the computation time between two PE's but also the communication time including circuit set up time and data transfer time. The same problem is propagated to the later steps, so we have the largest integration time for PE_{15} which is the bottleneck between convolution and backprojection stages. Therefore, though the average computation times are quite close to one another in both stages, the resulted convolution time is much longer than the averaged computation time in backprojection stage such that we do not still have high efficiency.

Another possible interpretation of why the efficiency improvement is limited is the following. Since it is usually difficult to have a combination (n_c, n_b) such that n_c/n_b is exactly equal to γ, there naturally exists an overhead due to unmatched processing time. For example, in Table 4.4., where the load distribution for the image size of 31 with combination (5, 1) is shown, each convolution PE is almost fully utilized. However, due to the mismatched processing time, the overall efficiency is still less than 90%. Also, as the computational load is distributed over more and more PE's, the communication load becomes relatively more dominant due to a smaller computational load and PE utilization would accordingly be lowered down even if a well-balanced load distribution is achieved.

Table 4.5. Load distribution for N=31 and $(n_c, n_b) = (28, 4)$ after load balancing

node	1st brdcst	brdcst	integrate	I/O	compute	node time	utilize
0	104	758	42	33	7300	7878	92.66%
1	104	101	604	0	7079	7892	89.70%
2	104	731	43	0	6983	7864	88.80%
3	104	27	796	0	6974	7906	88.21%
4	105	853	42	0	6855	7859	87.22%
5	105	550	428	0	6786	7873	86.19%
6	104	1011	43	0	6690	7851	85.21%
7	105	26	1030	0	6758	7921	85.32%
8	104	989	43	0	6713	7852	85.49%
9	105	689	441	0	6629	7867	84.26%
10	104	267	41	0	7518	7935	94.74%
11	104	26	618	0	7392	8143	90.78%
12	104	202	42	0	7587	7941	95.54%
13	104	26	626	0	7195	7955	90.45%
14	104	145	50	0	7692	7995	96.21%
15	105	28	1177	0	7856	9170	85.67%
16	105	328	43	0	7783	8263	94.19%
17	105	31	429	0	7708	8278	93.11%
18	105	324	42	0	7793	8269	94.24%
19	104	29	630	0	7703	8469	90.96%
20	105	458	43	0	7648	8257	92.62%
21	104	126	330	0	7710	8274	93.18%
22	104	649	43	0	7453	8253	90.31%
23	105	32	903	0	7739	8782	88.12%
24	101	398	41	0	7711	8256	93.40%
25	102	102	502	0	7561	8271	91.42%
26	102	450	44	0	7659	8259	92.74%
27	102	27	614	0	7811	8558	91.27%
31	535	905	43	0	8117	9604	84.52%
30	535	364	43	0	8682	9629	90.17%
29	535	944	597	0	8076	10155	79.53%
28	484	607	1091	45	8431	10664	79.06%

5. Conclusions

Based on the discussions in the preceding section, our observations made in this implementation may be summarized as follows:

- The simple task partitioning scheme for the convolution stage can work well for a large image size even if more than 20 PE's are employed. It can also work well for a small image size if the number of PE's is properly chosen.
- The computational load balancing is a crucial factor affecting the PE utilization and in turn the overall system performance.
- The computational load assignment is very important in minimizing the communication overhead. It is suggested that a heavier computational load is to be put in the later step in integration.
- Even if we can achieve a balanced load distribution, it does not guarantee that the overall efficiency will be high because it is still possible to have a substantial communication overhead due to potential time mismatching between two stages. Therefore, a proper combination should be employed for high efficiency. And if the time difference is not too large, it's also possible to distribute the load such that the mismatched time difference can be minimized and data flow is kept smoothly.

We have seen that it is feasible to efficiently parallelize the 3-D CT image reconstruction achieving a speed-up close to the numbers of PE's employed in most cases. The parallel computing system we used in this study is not necessarily tailored for this task. Therefore, it would be worthwhile to test other types of parallel machines, e.g., a shared memory architecture, and eventually to design a dedicated computing system for 3-D CT image reconstruction, which are our current research interests.

Acknowledgement

The authors would like to thank Dr. Y. S. Kim at Columbia University for his numerous helpful comments on the CT system model.

References

[1] Eiichi Tanaka, "Recent Progress on Single Photon and Positron Emission Tomography - from Detectors to Algorithms -," *IEEE Trans. on Nucl. Sci.*, vol. NS-34, No. 1, Feb 1987.

[2] F. Vermeulen, "An Improved Stochastic Reconstruction Technique for Tomographic Imaging," in *Proc. 2nd Int. Symp. Fundamentals of Tech. Progress in Medicine*, Liege, Belgium, Apr. 15-16, 1983.

[3] E. Tanaka, N. Nohara, T. Tomitani, and M. Yamamoto, "Utilization of Non-negativity Constraints in Reconstruction of Emission Tomography," in *Proceedings of the Ninth Conference of Information Processing in Medical Imaging*, Stephen L. Bacharach, Ed. June 10-14, 1985, Martinus Nijhoff, The Netherlands, 1985.

[4] S. Vishampayan, J. Stamos, R. Mayans, K. Koral, N. Clinthorne, and W. L. Rogers, "Maximum Likelihood Image Reconstruction for SPECT," *J. Nucl. Med.*, vol. 26, p20, 1985.

[5] D. G. Politte, "Reconstruction Algorithms for Time-of - flight Assisted Positron-emission Tomography," M.S. thesis, Washington University, St. Louis, MO, 1983.

[6] Eiichi Tanaka, "A Fast Reconstruction Algorithm for Stationary Positron Emission Tomography Based on a Modified EM Algorithm," *IEEE Trans. Med. Imaging*, vol. MI-6, No. 2, pp. 98-105, June 1987.

[7] T. M. Peters, "Algorithms for Fast Back-and-Re Projection in Computed Tomography," *IEEE Trans. Nucl. Sci.*, vol. NS-28, No. 4, pp. 3641-3647, Aug. 1981.

[8] Z. H. Cho, C. M. Chen and S.-Y. Lee, "Incremental Algorithms: Fast Backprojection Schemes for Parallel Beam Geometries," to appear in *IEEE Trans. on Med. Imaging*.

[9] C. J. Thompson and T. M. Peters, "A Fractional Address Accumulator for Fast Backprojection," *IEEE Trans. Nucl. Sci.*, vol. NS-28, No. 4,pp. 3648-3650, Aug. 1981.

[10] *Modular Image Processor (MIP), IP-300 Technical Manual*, Analogic Corporation, Wakefield, MA, 1983.

[11] R. Hartz, D. Bristow, and N. Mullani, "A Real-Time TOFPET Slice-Backproject Engine Employing Dual Am 29116 Microprocessors," *IEEE Trans. Nucl. Sci.*, vol. NS-32, No.1, pp. 839-842, Feb. 1985.

[12] W. F. Jones, L. G. Byars, and M. E. Casey Computer Technology and Imaging, Inc., "Positron Emission Tomographic Images and Expectation Maximization : A VLSI Architecture for Multiple Iterations Per Second," *IEEE Trans. Nucl. Sci.*, vol. 35, No. 1, pp. 620-624, Feb. 1988.

[13] J. Llacer and J. D. Meng, "Matrix-Based Image Reconstruction Methods for Tomography," *IEEE Trans. Nucl. Sci.*, vol. NS-32, No.1, pp. 855-864, Feb. 1985.

[14] S.-Y. Lee and J. K. Aggarwal, "Exploitation of Image Parallelism via The Hypercube," *Second Conference on Hypercube Multiprocessors*, Knoxville, TN, Sept. 1986.

[15] J .G. Rogers, R. Harrop and P.E. Kinahan, "The Theory of Three-dimensional Image Reconstruction for PET," *IEEE Trans. Med. Imaging*, vol. MI-6, No.3, pp. 239-243, Sept. 1987.

[16] J. B. Ra, C. B. Lim, and Z. H. Cho, "A New True Three-dimensional Reconstruction Algorithm for The Spherical Positron Emission Tomography (S-PET)," *Phys. Med. Biol.*, vol. 27, no. 1, pp. 37-50, 1982.

[17] Z. H. Cho, J. B. Ra, and S. K. Hilal, "True Three-dimensional Reconstruction (TTR) — Applications of Algorithm toward Full Utilization of Oblique Rays," *IEEE Trans. Med. Imaging*, vol. MI-2, pp. 6-18, 1983.

[18] J. B. Ra and Z.H. Cho, "Generalized True Three-Dimensional Reconstruction Algorithm," *IEEE Proceedings*, vol. 69, No. 5, May 1981.

[19] L. A. Shepp and B. F. Logan, "The Fourier Reconstruction of A Head Section," *IEEE Trans. Nucl. Sci.*, vol. NS-21, pp. 21-43, 1974.

[20] C. M. Chen, "Sequential and Parallel Approaches to Fast Image Reconstruction in 3-D Computerized Tomography," M.S. Thesis, Cornell University, Ithaca, New York, August 1989.

[21] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *Research Report* YALEU/DCS/RR-389, Yale University, June 1985.

[22] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison wesley, 1982.