The International Journal of Parallel, Emergent and Distributed Systems,
Vol. 21, No. 5, October 2006, 365–380

Taylor & Francis
Taylor & Francis Group

Parallel ART for image reconstruction in CT using processor arrays

DAN GORDON*

Department of Computer Science, University of Haifa, Haifa 31905, Israel

(Received 5 November 2004; in final form 26 January 2006)

Algebraic Reconstruction Technique (ART) is a widely-used iterative method for solving sparse systems of linear equations. This method (originally due to Kaczmarz) is inherently sequential according to its mathematical definition since, at each step, the current iterate is projected toward one of the hyperplanes defined by the equations. The main advantages of ART are its robustness, its cyclic convergence on inconsistent systems, and its relatively good initial convergence. ART is widely used as an iterative solution to the problem of image reconstruction from projections in computerized tomography (CT), where its implementation with a small relaxation parameter produces excellent results. It is shown that for this particular problem, ART can be implemented in parallel on a linear processor array. Reconstructing an image of n pixels from $\Theta(n)$ equations can be done on a linear array of $p = O(\sqrt{n})$ processors with optimal efficiency (linear speedup) and $O(n/p)$ memory for each processor. The parallel technique can be applied to various geometric models of image reconstruction, as well as to 3D reconstruction with spherically symmetric volume elements, using a 2D rectangular mesh-connected array of processors.

Keywords: ART; Computerized tomography; Kaczmarz; Image reconstruction; Linear equations; Sparse systems

1. Introduction

We consider the problem of using a processor array to obtain a parallel implementation of a certain algorithm for solving the problem of image reconstruction from projections in transmission computerized tomography (CT). This image reconstruction problem, after a discretization of the image domain, leads to a system of linear equations which is inevitably inconsistent, due to noise, inaccurate measurements, and the discretization process. Generally, image reconstruction methods can be divided into two types: transform methods and iterative methods. Transform methods are generally faster and can, in part, be implemented in hardware. The iterative methods are software-based, slower, but produce better quality images, even under adverse conditions such as noise or lack of sufficient data.

Algebraic Reconstruction Technique (ART) was one of the first iterative approaches to the problem of image reconstruction from projections in CT; see Herman [1]. In this method, originally due to Kaczmarz [2], each iterate is projected onto a hyperplane (defined by one of the equations). The hyperplanes are chosen sequentially, and the process continues in a cyclic

*Corresponding author. Email: gordon@cs.haifa.ac.il

The International Journal of Parallel, Emergent and Distributed Systems
ISSN 1744-5760 print/ISSN 1744-5779 online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/17445760600604157

manner until convergence. ART is often used with a relaxation parameter which controls the distance of the projection, and it achieves excellent results on CT data when used with a small relaxation parameter [1, sec. 11.5]. ART has been studied very extensively, both theoretically and experimentally. For results related to the convergence properties of ART with relaxation, for consistent and inconsistent systems, see [3–7].

The inherent sequentiality of ART is a hindrance to its efficient parallel implementation. However, for the specific problem of image reconstruction from projections in CT, ART can be executed so that entire "blocks" of projections can be done simultaneously instead of sequentially. The simultaneous projections in each such block are mathematically equivalent to performing the projections in any sequential order, so the resulting algorithm is also ART. This method of parallelizing ART is not new. For example, Bramley and Sameh [8] used such ideas for solving linear systems of equations obtained from partial differential equations. However, in such systems, the number of nonzero elements in each equation is a small constant, whereas in the image reconstruction problem, it is of the order $\sqrt{n}$, where n is the number of variables.

Our parallelization of ART uses linear and rectangular arrays of processors, also known as meshes. In these simple parallel models of computation, each processor is connected directly to only a small and fixed number of neighboring processors. Such computation models have been studied very extensively and they have many applications in sorting [9], computational geometry [10], and combinatorial and numerical algorithms [11].

If the image has n pixels and the number of equations is $\Theta(n)$, then, using a linear array of $p = O(\sqrt{n})$ processors, optimal efficiency (linear speedup) can be achieved. Furthermore, the memory requirement for each processor is $\Theta(n/p)$. We shall refer to the parallel implementation of ART on a linear processor array as PART. PART can be applied to various geometric models of image reconstruction, and it can be extended to a fully three-dimensional (3D) image reconstruction using spherically symmetric volume elements (called "blobs") instead of voxels [12–14]. The 3D reconstruction uses a 2D rectangular mesh-connected array of processors.

One of the problems incurred in parallel iterative techniques is that of the communication overhead. After each of the processors has done its allotted work, it is usually necessary to distribute the new values to other processors before the next iteration can begin. This communication problem often results in degraded performance (per processor) as the number of processors increases. In our implementation, the communication time is linear in the computation time.

We have simulated PART on SNARK93 [15] (a software package for image reconstruction), and verified that it performs no worse than the regular ART. It is quite straightforward to implement PART on an SIMD linear processor array such as the SliM-II [16] or the Sarnoff Engine [17]. Another potential implementation can utilize a linear array of PC's, each connected to its two neighbors by dedicated high-bandwidth communication cards.

One should note that the simple sequential ordering of the projections in ART is not optimal. Herman and Meyer [18] have shown that an ordering based on a certain prime number decomposition yields better results. For a recent study of several different orderings of the projections in ART see Kazantsev *et al.* [19]. It follows from these studies that the ordering of the projections used by PART is most probably not optimal, but this is more than offset by the parallelism of the method.

The rest of this paper is organized as follows: Section 2 presents the image reconstruction problem in CT and Section 3 presents some previous work on parallelization of image

reconstruction algorithms. Section 4 explains ART and a general parallel version of ART. Section 5 details the implementation of ART for image reconstruction on a linear processor array. Section 6 extends PART to various geometric models and to 3D reconstruction with blobs using a 2D array of processors. Section 7 presents simulation results, and the last section concludes with some further research directions.

2. Image reconstruction in transmission CT

In transmission CT, X-rays are passed through a cross-section of an object and readings are taken by detectors at the opposite side. Different parts of the cross-section attenuate the X-rays differently, resulting in different readings at the detectors. The problem is to reconstruct the attenuation map–or image–of the cross-section from the detector readings. Mathematically, the unknown attenuation is a nonnegative function of two variables defined on the cross-section. The most elementary method for reconstructing the attenuation is to overlay the cross-section with a Cartesian grid of pixels, and to consider the attenuation to be constant inside every pixel. Let n denote the total number of pixels.

In a typical setup, we assume that a line of parallel X-ray sources are aimed at a line of detectors, as shown in figure 1. X-rays are cast toward the detectors, which measure the attenuation along the rays' paths. The orientation of the sources and detectors is then rotated by some small angle, and the process repeats until the entire cross-section has been "covered" by equiangular orientations. We denote by m the total number of rays obtained in this manner.

Note that after a rotation of 180° the sources and the detectors have swapped their position with respect to the initial angle, but mathematically, the line integrals are the same as for the initial angle.

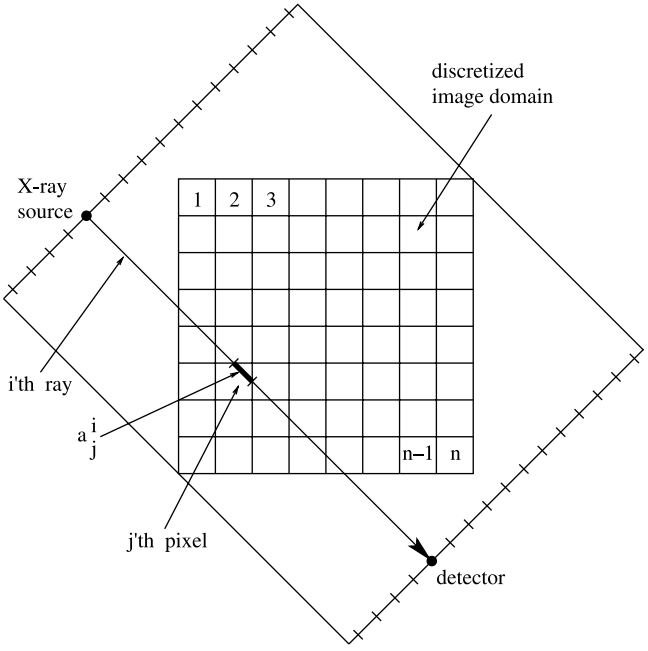

Figure 1. Tomographic image reconstruction in the discretized model.

We assume that the pixels and rays are numbered consecutively in some manner from 1 to n and from 1 to m, respectively. Let x_j denote the (assumed constant) value of the attenuation function in the jth pixel. It is assumed that the X-ray sources and detectors are points, and that the rays between them are lines. It is further assumed that the length of intersection of the ith ray with the jth pixel, denoted by a_j^i, represents the contribution of the jth pixel to the total attenuation along the ith ray.

The physical measurement of the total attenuation along the ith ray, denoted by b_i, is the line integral of the unknown attenuation function along the path of the ray. In the discretized setup, each line integral is approximated by a finite sum, so the (discretized) attenuation function is the solution of a system of linear equations

$$\sum_{j=1}^{n} a_j^i x_j = b_i \quad \text{for} \quad 1 \le i \le m, \quad \text{or, in matrix form:} \quad Ax = b. \tag{1}$$

The above system of equations is invariably inconsistent, due to inaccurate measurements, ray scattering, and the very discretization procedure.

3. Parallel algorithms for image reconstruction

The issue of parallelism in the context of image reconstruction in CT is by no means new. Since a complete overview of the literature on this topic is beyond the scope of this paper, we present only a sample. A prime source for parallel algorithms for optimization can be found in Censor and Zenios [20]. Three parallel implementations of the block-iterative MART algorithm on a Cray vector supercomputer can be found in [20, sec. 14.4, 15.3]. The approach there is based on control parallelism and utilizes the fast vector operations of the Cray.

Schmidt *et al.* [21] have implemented the filtered backprojection algorithm for image reconstruction on the Systola 1024 systolic array. Their method implements the fast fourier transform on an MIMD array of 1024 processing elements, each with a very limited local memory. Although filtered backprojection is in common use and it is fast, iterative algorithms are advantageous under adverse conditions such as noise and lack of sufficient data.

Censor *et al.* [22] introduced the component averaging (CAV) algorithm which is simultaneous in the same way as Cimmino's algorithm [23]. CAV can be parallelized in block-parallel mode; i.e. the equations are partitioned into blocks which are then operated on in parallel. The results from the different processors are then combined to form the next iterate. CAV was further extended to BICAV [24], which operates in block-sequential (also called "block-iterative") parallel mode.

Statistical methods for image reconstruction have proved to be very useful for both emission and transmission CT. The long reconstruction time of such methods have prompted several studies on parallelization. Johnson and Sofer [25] present a data-parallel approach to parallelizing the ML-EM algorithm for emission tomography. Their method utilizes symmetries of the system matrix and can be extended to other algorithms, including various block-iterative algorithms, such as ordered subsets expectation maximization (OSEM). They note, however, that although their approach can be applied to the variable-block ART [20, sec. 10.4], it cannot be used for the regular ART due to ART's inherently sequential nature.

Johnson *et al.* [26] present a parallel implementation of interior-point techniques for 3D PET reconstruction, again using the statistical approach of maximum likelihood (ML)

optimization. Their technique requires a global summation step after every backprojection. In Ref. [27], Johnson and Sofer extend their data-parallel approach to the parallel implementation of the primal-dual algorithm for obtaining the ML solution for emission CT.

Kole and Beekman [28] have studied the parallelization of the ordered subsets convex (OSC) algorithm of Kamphuis and Beekman [29] on a shared memory machine; they find that nearly linear speedup can be obtained with up to 40 processors. If more processors are added, performance (per processor) will probably begin to degrade due to the communication overhead. Several additional relevant references can be found in [28] for parallel image reconstruction in general, and transmission CT in particular.

4. ART and parallel ART

For $1 \le i \le m$, we denote by a^i the ith row vector of the matrix A in equation (1), and by $\langle u, v\rangle$ the dot product of two vectors u, v. $\mathbb{R}^n$ denotes the n-dimensional Euclidean space. Each equation of (1) corresponds to a hyperplane of $\mathbb{R}^n$. ART can be described as follows: starting from an arbitrary point $x^0 \in \mathbb{R}^n$, the kth iterate x^k is projected toward the next hyperplane, and the hyperplanes are chosen in cyclic order. To simplify our notation, we denote by $i(k)$ the kth index taken cyclically from 1 to m; i.e. $i(k) = (k \bmod m) + 1$. Furthermore, we assume for the sake of simplicity that the equations are normalized; i.e. for $1 \le i \le m$, the ith equation is divided by $\|a^i\| = \sqrt{\langle a^i, a^i\rangle}$. The ART algorithm, with a fixed relaxation parameter λ, is the following.

ALGORITHM 1(ART):. Initialization: $x^0 \in \mathbb{R}^n$ is arbitrary.

Iterative Step: Given x^k compute

$$x^{k+1} = x^k + \lambda\big(b_{i(k)} - \langle a^{i(k)}, x^k\rangle\big)a^{i(k)}. \tag{2}$$

Note that the algorithm does not specify a particular order between the equations; any cyclic order will do. This point will be used in our parallel implementation. Consider the application of equation (2) to one equation. Clearly, the only components of $x^k = \big(x_1^k, \ldots, x_n^k\big)$ which change their value as a result of this step are those whose corresponding coefficients in the equation are nonzero. In other words, we can replace equation (2) by

$$\text{For} \quad 1 \le j \le n, \quad x_j^{k+1} = \begin{cases} x_j^k & \text{if} \quad a_j^{i(k)} = 0, \\ x_j^k + \lambda\big(b_{i(k)} - \langle a^{i(k)}, x^k\rangle\big)a_j^{i(k)} & \text{otherwise} \end{cases} \tag{3}$$

Let S be a subset of the equations of (1). We define S to be independent if, for every $1 \le j \le n$, there is at most one equation in S with a nonzero coefficient of x_j. If S is independent then in $\mathbb{R}^n$, all the vectors formed from the coefficients of the equations of S are pairwise orthogonal. If we apply the sequence of projections of ART to S, the result will not depend on the order in which the projections are done. Furthermore, we can even do all the projections in parallel, and take the result to be the following: for $1 \le j \le n$, if all the coefficients of x_j in the equations of S are zero, then x_j^k is unchanged; otherwise, x_j^k takes the value obtained by applying the projection to the single equation in S in which the coefficient of x_j is nonzero.

Suppose now that the equations of (1) are partitioned into disjoint independent sets S_1, S_2, ... ,S_t. Denote by $\ell(k)$ the kth index taken cyclically from 1 to t; i.e. $\ell(k) = (k \bmod t) + 1$. Also, for every $1 \le j \le n$ and $1 \le \ell \le t$, denote by $i(j, \ell)$ the index of the single equation in S_ℓ in which the coefficient of x_j is nonzero; if no such equation exists, set $i(j, \ell) = 0$. The general parallel ART, with a fixed relaxation parameter λ is the following.

ALGORITHM 2 (GENERAL PARALLEL ART):. Initialization: $x^0 \in \mathbb{R}^n$ is arbitrary.

Iterative Step: Given x^k compute, for $1 \le j \le n$,

$$x_j^{k+1} = \begin{cases} x_j^k & \text{if} \quad i(j, \ell(k)) = 0, \\ x_j^k + \lambda\left(b_{i(j,\ell(k))} - \langle a^{i(j,\ell(k))}, x^k\rangle\right) a_j^{i(j,\ell(k))} & \text{otherwise} \end{cases} \tag{4}$$

Algorithm 2 can be applied in any setting where the equations can be partitioned into independent sets, and this is possible only when the system matrix A in equation (1) is sparse. As mentioned in the introduction, this formulation of ART has been used before for solving partial differential equations, where the number of nonzero elements in each row of the system matrix is a small constant.

5. Parallel ART for CT on a linear processor array

In this section we provide a detailed implementation of algorithm 2 on a linear processor array. We shall refer to this particular implementation as PART.

5.1 Preliminaries and overview

Recall that m is the total number of equations (rays) and n is the number of variables (pixels). If we denote by r the number of rays in one projection, then the number of projection angles is m/r. We make the following assumption about the geometric setup: both r and m/r are $\Theta(\sqrt{n})$ (i.e. for some constants $c_1, c_2 > 0$, $r = c_1\sqrt{n}$ and $m/r = c_2\sqrt{n}$). It follows from these assumptions that $m = \Theta(n)$. These assumptions are meaningful in the context of analyzing the time and space requirements of our implementation as functions of n.

For the purpose of implementing algorithm 2, we obtain independent sets of equations as follows. Consider the configuration of rays shown in figure 2. The figure represents all the rays of one projection, numbered consecutively from 1 to r as shown. Assume w.l.o.g. that the length of a pixel side is 1, and assume that the distance between adjacent rays in one projection, denoted by d, satisfies the following inequalities:

$$\sqrt{2}/2 \le d \le 1. \tag{5}$$

We shall explain later how to handle the case when the above inequalities are not satisfied. The right inequality ensures that for every projection angle, every pixel is intersected by at least one ray, even when the rays are parallel to one of the main axes. (If $d = 1$, we assume that the geometric configuration is such that the rays never coincide with any pixel boundaries.) At every stage of our algorithm, every pixel value will be stored in some processor which handles a ray passing through the pixel, so the right inequality ensures that at all times, every pixel value is stored in the memory of some processor. These pixel values

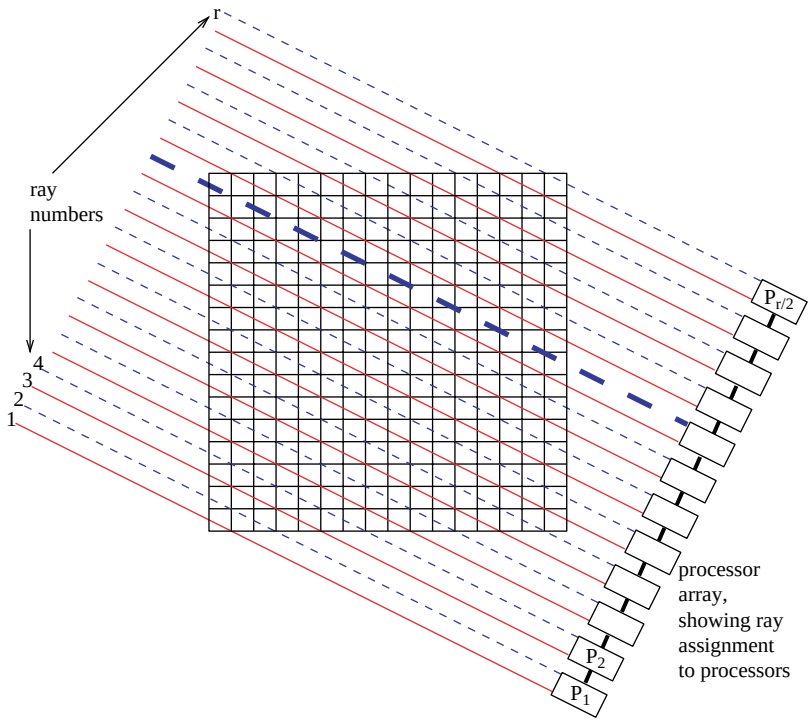

Figure 2. Partitioning rays into even and odd, and assignment to processors. (Colour version available online.)

(the x^k of ART) will always be distributed among the processors in such a way that the memory required for this purpose by each processor is $O(\sqrt{n})$; no external memory for the image will be used.

The left inequality ensures that two rays with odd numbers cannot share a pixel, and the same goes for rays with even numbers. This is shown in figure 2: The odd rays are red/solid and the even rays are blue/dashed. One blue/dashed ray is enhanced, and it can be seen that it shares no pixel with any other blue/dashed line. It follows that the set of odd rays of one projection corresponds to an independent set of equations, and the same goes for the even rays. This property is true for all the projection angles, and it forms the basis of our parallel implementation.

Assume for simplicity that r (the number of rays in one projection) is even and consider a processor array of $p = r/2$ processors, indexed consecutively from 1 to p. Later, we will consider the case $p < r/2$. We assign all the rays to be handled by the processors in consecutive manner as shown in figure 2. Each processor is assigned to two equations (or two rays) in the projection as follows: Processor 1 handles rays 1 and 2, processor 2 handles rays 3 and 4, and so on. For each projection angle, each processor will compute the coefficients of the equations that it handles; this can be done based on the information about the geometry of the problem.

5.2 Detailed algorithm

The parallel implementation proceeds as follows: initially, the geometric data are broadcast to all the processors. This data includes the image size, the inter-ray distances, the initial angle and the step angle. From this data, each processor should later be able to calculate the coefficients of the equation corresponding to its assigned rays, but these coefficients are not

all computed in advance: They are computed for the initial angle only, and after each rotation, the required coefficients for the new equations are computed only when they are needed. The new coefficients replace the previous coefficients in the processor's memory. This ensures that for each processor, the memory required for the coefficients is $O(\sqrt{n})$, since each ray intersects $O(\sqrt{n})$ pixels.

Next, all the b_i's of equation (1), for the initial angle only, are broadcast to all the processors. Based on the geometric information, each processor retains only the b_i's that it will use. For each new angle, the b_i's are broadcast as they are needed. This broadcast can be done in a "communication hiding" mode; i.e. concurrently with the processing of the equations of the previous angle.

As mentioned, each processor stores only the x_j^k's of the pixels intersected by the rays that it handles. For each new rotation angle, the processor will transmit (some of) these values to its adjacent processors and receive some new values from its adjacent processors. The new values will replace (in the processor's memory) those values which are not needed for the new angle, so the number of variables in its memory will remain $O(\sqrt{n})$. Thus, the total memory requirement for each processor (coefficients and variables) is also $O(\sqrt{n})$. Initially, each processor assigns the starting values for its variables (0 in our implementation).

The basic operation for each projection angle proceeds in two steps.

Step1: Each processor calculates the change in the values of the variables corresponding to the odd ray assigned to it (red/solid in figure 2).

Step 2: Each processor calculates the changes of all the variables corresponding to the even ray assigned to it (blue/dashed in figure 2). However, before it can do that, it needs to have the values of these pixels obtained by step 1. Some of these values are already in the processor's memory, but some are not. Figure 2 shows that every pixel intersected by an even ray (blue/dashed line) can be one of three types:

1. The pixel is intersected only by the even (blue/dashed) line and not by any of its adjacent (red/solid) lines.
2. The pixel is intersected by an odd (red/solid) line handled by the same processor.
3. The pixel is intersected by an odd (red/solid) line handled by its higher-indexed neighboring processor.

In cases 1 and 2, the pixel value is already available to the processor. In case 3, the pixel value is obtained by an intermediate step in which each processor sends such values to its lower-indexed neighbor, and all the transfers are done in parallel. Note that the number of values that need to be transferred is of the same order as the number of variables handled by the processor; i.e. $O(\sqrt{n})$. Hence, the communication time is of the same order as the computation time. Step 2 now proceeds with all the even rays.

Steps 1 and 2 complete all the calculations corresponding to one projection angle. The next two steps will correspond to the next projection angle, which is obtained by rotating all the rays by some small angle. This is shown in figure 3: the red/solid lines correspond to a projection that was already calculated, and the enhanced blue/dashed line is the rotation of the enhanced red/solid line. If the rotation angle is sufficiently small, then all the pixels intersected by the blue/dashed line are intersected by the enhanced red/solid line or by one of its neighboring red/solid lines. This means that all the pixel values required by a processor for the new angle are either already held by the processor or they are available from its nearest

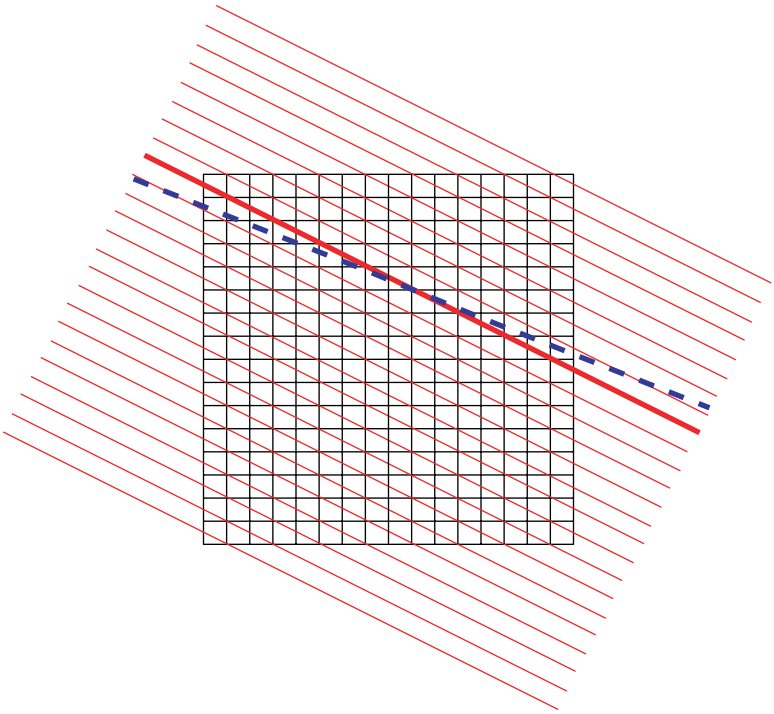

Figure 3. Rotation of the rays. (Colour version available online.)

neighbors. The transmission of such data can be carried out in parallel and takes time $O(\sqrt{n})$. If the angle of rotation is not small enough, then the necessary pixel values are still available from a small neighborhood of processors whose size depends on the geometry, so the time to transfer the data is also $O(\sqrt{n})$.

After a rotation of 180°, the processors will handle rays in the same configuration as the initial one, but now rays 1 and 2 will be handled by processor $r/2$, and so on. Thus, after one iteration through all the rays, the information is available in the processors for the next iteration. This continues for as many iterations as required. At the end, we need to output all the data from one end of the array. This is done in $r/2$ steps, where at each step, each processor moves $\Theta(\sqrt{n})$ data to its predecessor, and the first processor outputs the results.

5.3 Additional comments and main result

Some comments concerning the assumed inequalities of the inter-ray distance d; see equation (5). If $d < \sqrt{2}/2$, then for some integer $k > 2$, we need to assign k rays to each processor. k is taken as the smallest integer such that $kd \geq \sqrt{2}$; i.e. $k = [\sqrt{2}/d]$. PART now proceeds as explained above, except that for each projection, we have k steps instead of the previous two. As for the case $k = 2$, between every two steps, each processor needs to pass image data to its lower-indexed neighbor. Note that if $d < \sqrt{2}/2$ then the right inequality of equation (5) is also satisfied.

We also assumed in equation (5) that $d \leq 1$, and we need to consider the situation when $d > 1$. In this case, for some projection angles, there will be some pixels which are not intersected by any ray of the projection; just consider a projection direction parallel to the

x- or y-axis. Recall that $d \leq 1$ guarantees that for every projection angle, every pixel is "covered" by some ray of the projection, so all the image data is always stored in the memories of the processors. If $d > 1$, we need to ensure that no image data gets "lost" when some pixel is not covered by any ray in a certain projection angle. This problem can be solved by considering the underlying geometry: for every projection angle, every processor can determine if there is any pixel which falls between its allotted rays or between its highest-indexed ray and the first ray of its successive processor. If so, the processor maintains the value of this pixel, and this includes the transfer of such pixel values to adjacent processors as needed between steps (such as steps 1 and 2) or between the processing of two successive projection angles.

We found that the time required for each projection angle (for computations and communications) is $O(\sqrt{n})$. Hence, the time for one entire sweep of ART (dealing with all m equations) takes $O(\sqrt{n}m/r) = O(n)$ time (recall that r and m/r are both $\Theta(\sqrt{n})$). Also, from the above description of the final step, it follows the time to output the result is $O(r\sqrt{n}) = O(n)$, which is of the same order as one sweep of ART.

If the number of processors is p and $p < r/2$, then we can assign $k = [r/p]$ rays per processor and the time for one projection would take $O(k\sqrt{n}) = O(r\sqrt{n}/p) = O(n/p)$ steps. The memory required for each processor is $O(k\sqrt{n}) = O(n/p)$. The time for one entire sweep of ART will now be $O(mn/rp) = O(n\sqrt{n}/p)$. The time to output the results at the end will be p steps, where at each step, $O(n/p)$ data is transmitted from each processor to its lower-indexed neighbor, and from the first processor to the outside. Hence, the time to output the results is the same as for the case $p = r/2$, namely, $O(n)$. The following theorem summarizes our parallel implementation of ART.

THEOREM 5.1. Given a reconstruction problem with an image of n pixels and assuming that both the number of projection angles and r (the number of rays per projection) are $\Theta(\sqrt{n})$, then PART can be carried out in $O(n/p)$ time per projection angle on a linear processor array of $p \leq r/2$ processors; i.e. optimal efficiency (linear speedup) in the execution of ART can be achieved. The memory required for each processor is $O(n/p)$, and the time to output the results at the end is $O(n)$.

5.4 Some comments on implementation and enhancements

The runtime of any actual implementation would be greatly enhanced by the addition, to each processor, of a dedicated array processor. The array processor would be used in equation (4) for both the fast evaluation of the dot product and for updating the variables x_j^k handled by the processor.

At each basic step, each processor has to update $O(\sqrt{n})$ variables, and this takes time $O(\sqrt{n})$. In theory, this can be speeded up by replacing the processor array with a modified mesh of trees; see [11]. Every pair of rays is now handled by one row of the mesh, and the dot product of equation (4) is evaluated in logarithmic time by the binary tree attached to the row. In a regular mesh of trees, there are also trees attached to each column, but this is not needed for our application. Transferring data from one row to another can be done in $O(1)$ time. The time requirement would now be $\Theta(\log n)$ per projection. Note that the efficiency of this setup is not optimal, because the number of processors is multiplied by a factor of $\sqrt{n}$, but the reduction in time is from $\Theta(\sqrt{n})$ to $\Theta(\log n)$.

6. Extensions to other geometric models and to 3D reconstruction

6.1 Various two-dimensional geometric models

A most common geometrical setup is that of the fan-beam geometry, in which an X-ray source radiates rays in the shape of a fan and the detector provides readings for all the rays of the fan. It is well known that such a geometric setup can be translated to the parallel beam geometry by a process known as rebinning; see Herman [1]. This method associates every ray in a given fan beam with all the rays that are parallel to it in other fan beams. Parallel ART can now be applied.

In the basic model that we considered, we assumed that the rays form a line through the cross-section of the object. A different geometric model is obtained by considering each ray to be a strip of a certain width. In this application, each coefficient a_j^i is taken as the area of intersection of the ith strip and the jth pixel. PART is also applicable to this geometry, based on the same principle as before: for some integer $k > 2$, k consecutive rays are assigned to each processor. k is taken as the smallest integer such that, for every projection angle, any strips numbered i and $i + k$ do not share any pixel. The computation now proceeds as before.

Our basic model was based on the assumption that throughout a pixel, the attenuation value was constant. Lewitt [12,13] introduced a different mathematical model based on spherically symmetric volume elements, also known as "blobs". The basis function of the pixel (or voxel in 3D) model has a value of 1 inside a unit square (cube in 3D) and 0 outside. The blob basis function is radially symmetric with a value of 1 at the center and smoothly and monotonically decreasing to zero with the distance from its center. A blob has a certain specified radius beyond which its value is zero. Blobs are a generalization of the well known class of Kaiser–Bessel window functions used in digital signal processing. The radius of the blob is chosen so that the entire region of interest is covered, so the blobs necessarily overlap. See also Matej *et al.* [14].

Implementing PART in 2D using blobs is straightforward: as in the case of strips, for some integer $k \geq 2$, k consecutive rays are assigned to each processor. k is taken as the smallest integer such that, for every projection angle, two (parallel) rays numbered i and $i + k$ cannot both intersect a blob's positive area. This is done by taking $k = [D/d]$, where D is a blob's diameter and d is the inter-ray distance.

6.2 Three-dimensional PART with a rectangular mesh

A common model for 3D image reconstruction considers parallel slices and each slice is reconstructed independently from the other slices. In the voxel model, PART can be implemented with a rectangular mesh-connected array of processors, where each processor is connected to its four neighbors. Each column of the mesh can be assigned to one or more slices and perform the regular PART on its slice(s) independently of the other columns.

In the blob model, the columns of the mesh cannot operate independently because of the overlap of the blobs, and a modification is required. Each column of the array is assigned some $k \geq 2$ consecutive slices, where k is chosen as explained above for the 2D blob case. The idea is for each column of the mesh to handle its slices in succession, following the same principle along the slices as done in the 2D PART, with communications between the columns in order to transfer blob data. This is detailed below.

- Every column of processors will handle k consecutive slices.
- When a column (of processors) is processing a particular slice, every processor in the column will handle k consecutive rays as in the 2D case.

- Step 1 (along the slices): each column of processors performs one complete cycle of PART on its first slice exactly as in the 2D case. Note that the choice of k guarantees that two different columns will never operate on the same blob.
- Intermediate communication step: after the previous step, each column of processors sends required blob data to its lower-indexed neighboring column. This data transfer is done along the communication edges between row elements: every processor in column $j \geq 2$ holds the image data of k rays, and it sends the required data to its neighboring processor in the same row but with column index $j - 1$.
- Every column of processors now performs the slice-processing step on its next slice (same as step 1 above), and this continues until the last slice has been processed. Note that now, one complete cycle of ART has been performed on all the equations of the system.
- The above process now repeats itself, but before performing the slice-processing step on the first slice again, another communication step is needed. Due to our choice of k, every column indexed ℓ (except the last) holds the blob data of all the slices of column $\ell + 1$. Column ℓ transfers the required blob data to column $\ell + 1$. This is also done along the communication edges between row elements.
- If the processor mesh is not large enough to allow k rays per processor (or k slices per column), then more rays (or slices) need to be alloted to each processor (or each column). Note that, as in the 2D case, a processor may now need to maintain extra image data.

7. Experimental results

We have used the SNARK93 [15] software package, in which many image reconstruction algorithms have been implemented, including ART. By modifying the function pick within SNARK93, we were able to change the order in which equations were handled so that the algorithm simulated the PART algorithm. Specifically, in every projection we first performed the ART projection on all the odd rays, and then on all the even rays. Since the sequential and parallel execution on independent sets are mathematically equivalent, our sequential simulation produced results identical to those of the parallel algorithm.

Four different test cases were examined, all at a resolution of 255×255. Case 1 was the Herman head phantom [1, sec. 4.3], which is specified by a set of ellipses, with a specific attenuation value attached to each elliptical region. The pixel size was set identical to the inter-ray distance; this setup ensures the least amount of artifacts in the reconstructed images (as can be readily verified by experimentation). The reconstruction used 180 equidistant angles, resulting in a system of 65,025 variables and 64,980 rays—an almost exactly determined system.

Case 2 was based on the same phantom, but the number of equations is (approximately) 25% of their number in case 1. This was obtained by doubling both the inter-ray distances and the angle between successive projection angles. The resulting system of equations is thus strongly underdetermined. Such a setup is compatible with a situation of low dose radiation, as may be mandated by clinical requirements.

Cases 3 and 4 were obtained from 1 and 2, respectively, by adding noise to the readings. The noise was added by multiplying each of the b_i s of equation (1) by a random number chosen from a Gaussian distribution with a mean of 1.0 and standard deviation of 0.05. Note that we are experimenting with transmission CT, so our experiment with noise differs from the accepted practice in emission CT. Table 1 summarizes the four test cases.

Table 1. The four different test cases (resolution 255 × 255).

Case	*Variables*	*Equations*	*Projections*	*Rays*	*Noise*
1	65,025	64,980	180	361	No
2	65,025	16,290	90	181	No
3	65,025	64,980	180	361	Yes
4	65,025	16,290	90	181	Yes

The values of b_i, $1 \leq i \leq m$, are calculated by computing the line integrals through the elliptical regions (without reference to the discretization). Thus, the system (1) is basically inconsistent, because the left-hand-side is only an approximation to the actual integrals. This matches the real-life situation where the b_i's are actual X-ray readings through an object but the region of interest is discretized as above. Both cases were run for 40 iterations. We use the term iteration to refer to a single whole sweep through all equations of the system.

Two measures were used to define how much a reconstructed image diverges from the phantom, the distance and the relative error. These are calculated by SNARK93 [15], and defined as follows. Let $\hat{x}$ denote the phantom and x^k denote the reconstructed image after k iterations. Let S denote the set of indices j of pixels which are in the region of interest and let α be the number of elements in S. The average value of $\hat{x}$ is defined by

$$\hat{\rho} = \frac{1}{\alpha} \sum_{j \in S} \hat{x}_j,$$

and the standard deviation of $\hat{x}$ is defined as

$$\hat{\sigma} = \sqrt{\frac{1}{\alpha} \sum_{j \in S} (\hat{x}_j - \hat{\rho})^2}.$$

The distance between x^k and $\hat{x}$ is defined as

$$\delta_k = \frac{1}{\hat{\sigma}} \sqrt{\frac{1}{\alpha} \sum_{j \in S} (x_j^k - \hat{x}_j)^2}.$$

Setting $\hat{\tau} = \sum_{j \in S} |\hat{x}_j|$, the relative error of x^k is defined by

$$\varepsilon_k = \frac{1}{\hat{\tau}} \sum_{j \in S} |x_j^k - \hat{x}_j|.$$

Tables 2 and 3 show the minimal distance and relative error measures for ART and PART, for Cases 1 and 2, together with the iteration numbers at which the minimum values were obtained. In both cases, the measures are not significantly different from each other.

Table 2. Case 1: minimal distance and relative error, and iteration number(s) at which they were obtained.

Measure	*ART*	*PART*
Distance	0.0807, iteration 12–13	0.0819, iteration 13
Relative error	0.0497, iteration 8–9	0.0531, iteration 11

Table 3. Case 2: minimal distance and relative error, and iteration number(s) at which they were obtained.

Measure	*ART*	*PART*
Distance	0.1825, iteration 34–40	0.1826, iteration 34–40
Relative error	0.1126, iteration 15–16	0.1132, iteration 16–17

As far as images were concerned, no differences between ART and PART could be discerned visually in any of the examples that we studied. Figure 4 shows the phantom and reconstructed images for Cases 1 and 2. Case 2 is shown after 10 and 20 iterations (in Case 1, there were no visible improvements after 10 iterations).

As to Cases 3 and 4, again, the differences in the measures obtained by ART and PART were negligible, and no differences could be seen between the reconstructed images. However, it should be noted that in all the cases we examined, the measures for the regular (sequential) ART were consistently very slightly better than the measures for PART. This is due to the fact that PART is equivalent to ART, but with a different order between the equations. As noted in the introduction, different orders of taking the projections yield different results.

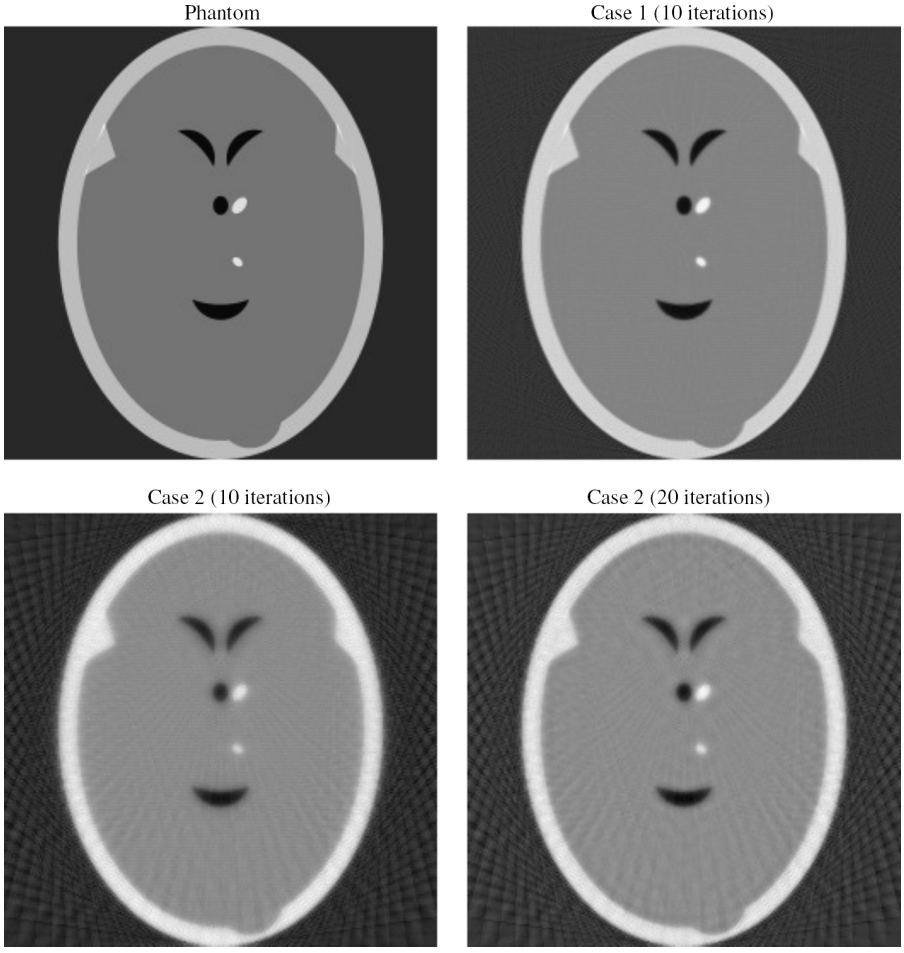

Figure 4. Phantom and reconstructed images for Cases 1 and 2.

8. Conclusions

This paper presented a technique whereby ART, applied to the problem of image reconstruction from projections in transmission CT, can be parallelized with optimal efficiency by using a linear processor array. This method puts ART on a more competitive standing as compared to transform methods, from a purely computational point-of-view. ART is already known to be superior to such methods with respect to the quality of reconstruction, especially when limited views are available, but its main problem has always been its inherent sequential nature.

Parallel ART can be applied to different types of geometric data such as fan-beam geometry and spherically symmetric volume elements (blobs) instead of voxels. Furthermore, it is easily extended to reconstruct a 3D image by utilizing a rectangular processor array, even with the blob model.

Future research on parallel ART should concentrate in two major directions. One is the extension to the spiral- or helical- geometry of modern CT appliances. Another, more practical direction, is the actual implementation, evaluation and testing of Parallel ART on linear and rectangular processor arrays.

Acknowledgements

The author is indebted to the anonymous reviewers whose comments led to a much-improved presentation.

References

[1] Herman, G.T., 1980, *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography* (New York: Academic Press).
[2] Kaczmarz, S., 1937, Angenäherte Auflösung von Systemen linearer Gleichungen, *Bulletin de l'Académie Polonaise des Sciences et Lettres*, **A35**, 355–357.
[3] Herman, G.T., Lent, A. and Lutz, P.H., 1978, Relaxation methods for image reconstruction, *Communication of the ACM*, **21**, 152–158.
[4] Tanabe, K., 1971, Projection method for solving a singular system of linear equations and its applications, *Numerische Mathematik*, **17**, 203–214.
[5] Eggermont, P.P.B., Herman, G.T. and Lent, A., 1981, Iterative algorithms for large partitioned linear systems, with applications to image reconstruction, *Linear Algebra and Its Applications*, **40**, 37–67.
[6] Trummer, M.R., 1981, Reconstructing pictures from projections: on the convergence of the ART algorithm with relaxation, *Computing*, **26**, 189–195.
[7] Censor, Y., Eggermont, P.P.B. and Gordon, D., 1983, Strong underrelaxation in Kaczmarz's method for inconsistent systems, *Numerische Mathematik*, **41**, 83–92.
[8] Bramley, R. and Sameh, A., 1991, Domain decomposition for parallel row projection algorithms, *Applied Numerical Mathematics*, **8**, 303–315.
[9] Akl, S.G., 1985, *Parallel Sorting Algorithms* (Orlando, Florida: Academic Press).
[10] Miller, R. and Stout, Q.F., 1996, *Parallel Algorithms for Regular Architectures: Meshes and Pyramids* (New York: Academic Press).
[11] Leighton, F.T., 1992, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* (San Mateo, California: Morgan Kaufman).
[12] Lewitt, R.M., 1990, Multidimensional digital image representations using generalized Kaiser-Bessel window functions, *Journal of the Optical Society of America A*, **7**, 1834–1846.
[13] Lewitt, R.M., 1992, Alternatives to voxels for image representation in iterative reconstruction algorithms, *Physics in Medicine and Biology*, **37**, 705–716.
[14] Matej, S. and Lewitt, R.M., 1996, Practical considerations for 3-D image reconstruction using spherically symmetric volume elements, *IEEE Transactions on Medical Imaging*, **15**, 68–78.
[15] Browne, J.A., Herman, G.T. and Odhner, D., 1993, SNARK93: A programming system for image reconstruction from projections, Technical Report MIPG198, The Medical Image Processing Group (MIPG), Dept. of Radiology, University of Pennsylvania.

[16] Chang, H., Lee, C. and Sunwoo, M.H., 1997, SliM-II: A linear array SIMD processor for real-time image processing. *Proceedings of the 1997 International Conference on Parallel and Distributed Systems (ICPADS '97)* Annual Conference, December (Los Alamitos, CA: IEEE Computer Society Press), pp. 132–137.
[17] Knight, S., Chin, D., Taylor, H. and Peters, J., 1992, The Sarnoff Engine: A massively parallel computer for high definition system simulation. *Proceedings of the 1992 International Conference on Application Specific Array Processors* Annual Conference, August (Los Alamitos, CA: IEEE Computer Society Press), pp. 342–356.
[18] Herman, G.T. and Meyer, L.B., 1993, Algebraic reconstruction techniques can be made computationally efficient, *IEEE Transactions on Medical Imaging*, **MI-12**, 600–609.
[19] Kazantsev, I.G., Matej, S. and Lewitt, R.M., 2005, Optimal orderings of projections using permutation matrices and angles between projection subspaces, *Electronic Notes in Discrete Mathematics*, **20**, 205–216.
[20] Censor, Y. and Zenios, S.A., 1997, *Parallel Optimization: Theory, Algorithms, and Applications* (New York: Oxford University Press).
[21] Schmidt, B., Schimmler, M. and Schröder, H., 2001, Tomographic image reconstruction on the instruction systolic array, *Computing and Informatics*, **20**, 27–42.
[22] Censor, Y., Gordon, D. and Gordon, R., 2001, Component averaging: an efficient iterative parallel algorithm for large and sparse unstructured problems, *Parallel Computing*, **27**, 777–808.
[23] Cimmino, G., 1938, Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari, *La Ricerca Scientifica XVI, Series II, Anno IX*, **1**, 326–333.
[24] Censor, Y., Gordon, D. and Gordon, R., 2001, BICAV: a block-iterative parallel algorithm for sparse systems with pixel-dependent weighting, *IEEE Transactions on Medical Imaging*, **20**, 1050–1060.
[25] Johnson, C.A. and Sofer, A., 1999, A data-parallel algorithm for iterative tomographic image reconstruction. *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation* Annual Conference (Los Alamitos, CA: IEEE Computer Society Press), pp. 126–137.
[26] Johnson, C.A., Seidel, J. and Sofer, A., 2000, Interior-point methodology for 3-d pet reconstruction, *IEEE Transactions on Medical Imaging*, **19**, 271–285.
[27] Johnson, C.A. and Sofer, A., 2001, A primal-dual method for large-scale image reconstruction in emission tomography, *SIAM Journal on Optimization*, **11**, 691–715.
[28] Kole, J.S. and Beekman, F.J., 2005, Parallel statistical image reconstruction for cone-beam x-ray CT on a shared memory computation platform, *Physics in Medicine and Biology*, **50**, 1265–1272.
[29] Kamphuis, C. and Beekman, F.J., 1998, Accelerated iterative transmission CT reconstruction using an ordered subset convex algorithm, *IEEE Transactions on Medical Imaging*, **17**, 1101–1105.