Distributed computation of linear inverse problems with application to computed tomography reconstruction

Yushan Gao,Thomas Blumensath

Abstract

The inversion of linear systems is a fundamental step in many inverse problems. Computational challenges exist when trying to invert very larger linear systems, where limited computing resources mean that only part of the system can be kept in computer memory at any one time. We are here motivated by tomographic inversion problems which often lead to linear inverse problems where millions if not billions of individual x-ray measurements are made and used to estimate x-ray attenuation in millions of locations. In state of the art x-ray systems, even a standard scan can produce 4 million individual measurements and the reconstruction of x-ray attenuation profiles typically requires the estimation of a million attenuation coefficients. This estimation is currently achieved using fast algorithms that operate on subsets of the data at a time. However, for these methods to work, simple scanning geometries have to be used. There is now significant interest in the use of more generic, iterative reconstruction methods, which tend to perform better especially when using more generic scan trajectories, when there is significant noise or when projection data are significantly under-sampled. However, to deal with the large data-sets encountered in real applications and to utilise modern GPU based computing architectures, parallel computing schemes are increasingly applied. For iterative x-ray tomographic reconstruction both row and column action methods have been proposed to utilise parallel computing architectures. However, individual computations in current methods need to know either the entire set of observations or the entire set of estimated x-ray absorptions, which can be prohibitive in many realistic big data applications.

This paper presents a novel CT image reconstruction algorithm which we call coordinate-reduced steepest gradient descent (CSGD). CSGD is fully parallelizable. By combing a row action method with the partial update scheme of a column action method, we derive a parallelizable algorithm where individual computations only require access to arbitrary subsets of the data and the estimated quantities. As this algorithm works with arbitrary subsets of the data and reconstructed volume, it does not rely on the use of standard CT scanning trajectories. To speed up the convergence of the method, we develop a non-homogeneously randomised selection criteria where projection data is selected stochastically depending on the subset of the reconstruction volume that is to be updated. In effect, this guarantees that sub-matrices of the system matrix are selected more frequently if they are dense and less likely if they are sparse, thus maximising information flow through the algorithm. A grouped version of the algorithm (Grouped CSGD (GCSGD)) is also proposed to further improve convergence speed and performance. Algorithm performance is verified experimentally.

Index Terms CT image reconstruction; parallel computing; gradient descent; coordinate descent ; linear inverse systems

I. INTRODUCTION

In transmission computed tomography (CT), standard scan trajectories, such as rotation based or helical trajectories, allow the use of efficient analytical reconstruction techniques such as the filtered backprojection algorithm (FBP) [29] and the Feldkamp Davis Kress (FDK) [28] method. However, in low signal to noise settings, if scan angles are under-sampled or if non-standard trajectories are used, then less efficient, iterative reconstruction methods can provide significantly better reconstructions [15], [34], [12]. These methods model the x-ray system as a linear system of equations:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e}, \tag{1}$$

where $\mathbf{y}, \mathbf{A}, \mathbf{x}$ and $\mathbf{e}$ are projection data, system matrix, reconstructed image vector and measurement noiss respectively. One drawback of iterative methods is however the relatively lower computational efficiency, which limits their use, especially in many x-ray tomography problems, where $\mathbf{y}$ and $\mathbf{x}$ can have millions of entries each [25] and where $\mathbf{A}$, even though it is a sparse matrix, can have billions of entries. For realistic data-set sizes, these methods thus require significant computational resources, especially as the matrix $\mathbf{A}$ is seldom kept in memory but is instead re-computed on the fly, which can be done relatively efficiently using the latest Graphical Processor Unit (GPU) based parallel computing platforms. However, as GPUs have limited internal memory, this typically requires the data $\mathbf{y}$ and/or reconstruction volume $\mathbf{x}$ to be broken into smaller subsets on which individual computations are performed. The development of efficient algorithms that only work on subsets of the data at any one time is thus becoming of increasing interest [18], [4].

Currently, most of these methods can be divided into two categories: *row action methods*, which operate on subsets of the observations $\mathbf{y}$ at a time and *column action methods*, which operate on subsets of the voxels $\mathbf{x}$ at a time [6], [35], [13]. Algorithms from both of these camps often also have parallelizable versions.

Row action methods: A classical method is the Kaczmarz algorithm (also known as the Algebraic Reconstruction Technique (ART)) which together with its block based variants are widely used in tomographic image reconstruction [12], [22]. When using a small relaxation parameter, this method achieves a weighted least square solution in overdetermined situation via iteratively solving each row of the system [23], [8]. Increased convergence rates are obtained with block Kaczmarz methods, which are also known as the Simultaneous Algebraic Reconstruction Technique (SART) [4]. By simultaneously selecting several rows of projection data, SART can also be converted into efficient parallel strategies [7]. Traditionally, block methods select sequential subsets or randomized subsets [24] or ordered subsets [23]. Some of these subset selection schemes require the calculation of the eigenvalue decomposition of the system matrix, which is currently not possible in CT reconstruction due to the size of the system matrix. Apart from the Kaczmarz method, another classical row action method is the simultaneous iterative reconstruction technique (SIRT). The SIRT method updates each reconstructed image voxel by combining all projection values whose corresponding x-rays passing through this voxel. In this way, the convergence rate significantly increases but so does the computation load [16], [33]. To lower the computational load within one iteration, the block form of SIRT is widely used in distributed computing systems [4], [3], yielding results of superior quality at the cost of increasing the reconstruction time.

Since in CT systems the system matrix is often large and sparse, component averaging (CAV) and its block form (BICAV) methods have been developed to utilise the sparsity property [11], [9], [10]. These methods differ from the classical Kaczmarz method, which sequentially projects the current iterate onto a single hyper-plane and requires a small relaxation parameter to achieve weighted least squares. CAV and BICAV methods simultaneously project the solution onto all hyper-planes and are guaranteed to converge to a weighted least-squares solution [19]. BICAV can use an additional coefficient matrix based on the number of non-zero elements in each column within a row block of the system matrix. This significantly increases the convergence rate compared to the original CAV method.

Column action methods: Column action methods are also called iterative coordinate descent method(ICD). They reduces the N-dimensional optimization problem into a one-dimensional problem, which is shown to have a faster convergence rate in the margins of the reconstructed image [5]. A non-homogeneous (NH-ICD) update strategy is proposed in [36], [37] to increase the convergence rate. The strategy first generates a pixel/voxel selection criterion (PSC) and then based on this criterion voxels which are furthest from convergence are selected to be updated. To ensure the PSC itself is sufficiently updated, the NH-ICD is often alternately used with homogeneous ICD. This method, however, is hard to parallelize because of its sequential nature and requires the scanning trajectory to be circular or helical. To increase the scalability and parallelism, a block form ICD (B-ICD) was proposed in [2]. The volume object is sliced along with the helical rotation axis and within one slice several neighbouring voxels are grouped together as a block. After all blocks within one slice are updated, the slice index steps along with the axial direction to the next slice and repeats the same iteration. Experiments show that a bigger block size is of help to increase the convergence rate, but at the cost of heavier computation amount. This is because this method does not use a separable surrogate function and thus makes the coefficient matrix into the inverse of a non-diagonal Hessian matrix. The Hessian matrix is dense when using trans-axial block and the inversion becomes increasingly difficult when the block size increases. To reduce the computation complexity brought by the non diagonal Hessian matrix, the ABCD algorithm, which is a derivative of the ICD method, combines the pixel line feature of [36], [37] and the block form [2] to update the pixel line simultaneously. The first improvement of the ABCD method is that it uses a separable quadratic surrogate function and thus the coefficient matrix is no longer the inverse matrix of a non-diagonal Hessian matrix but of a diagonal Hessian matrix, which means that the calculation of the coefficient matrix is far more efficient [2]. Another improvement is that it forms a pixel line as a block to decrease the coupling among voxels in one group. This partitioning leads to smaller diagonal elements in the Hessian matrix, thus a bigger step size is found when the Hessian is inverted, leading to a faster convergence rate. Inspired by NH-ICD, the NH idea is also applied to the ABCD method by more frequently updating the axial blocks which change by the largest amount during an iteration [21]. However, this block method is only suitable for standard CT scanners with circular or helical scanning trajectories and may not be appropriate for arbitrary scanning trajectories.

To optimize systems of linear equation $\mathbf{y} = \mathbf{A}\mathbf{x}$, [14] presents a concise summary of both row action and column action algorithm, which are show in Algo.1 and Algo.2.

Algorithm 1 Generic row action iteration

```
Initialization: $\mathbf{x}^0 \in \mathbb{R}^n$ is arbitrary. Both matrix $\mathbf{A}$ and projection data $\mathbf{y}$ are divided into $p$ row blocks. $\mathbf{A}_i$ and $\mathbf{y}_i$ are corresponding $i^{th}$ row blocks. $\mathbf{x}^k$ is the estimate or $\mathbf{x}$ in the $k^{th}$ iteration. $\omega_i$ and $\mathbf{M}_i$ are relaxation parameters and coefficient matrices respectively.
for $k = 0, 1, 2, ...$(epochs or outer iterations) do
    $\mathbf{z}^0 = \mathbf{x}^k$
    for $i = 1, 2, ..., p$ (inner iterations) do
        $\mathbf{z}^i = \mathbf{z}^{i-1} + \omega_i \mathbf{A}_i^T \mathbf{M}_i (\mathbf{y}_i - \mathbf{A}_i \mathbf{z}^{i-1})$
    end for
    $\mathbf{x}^{k+1} = \mathbf{z}^p$
end for
```

Algorithm 2 Generic column action iteration

```
Initialization: $x^0 \in \mathbb{R}^n$ is arbitrary. Both matrix $\mathbf{A}$ and vector $\mathbf{x}$ are divided into $q$ column blocks. $\mathbf{A}^j$ and $\mathbf{x}_j$ are corresponding $j^{th}$ column blocks. $\mathbf{r}^{0,1} = \mathbf{y} - \mathbf{A}\mathbf{x}^0$. $\omega_i$ and $\mathbf{M}_i$ are relaxation parameters and coefficient matrices respectively.
for $k = 0, 1, 2, ...$(epochs or outer iterations) do
    for $j = 1, 2, ..., q$(inner iterations) do
        $\mathbf{x}_j^{k+1} = \mathbf{x}_j^k + \omega_j \mathbf{M}_j (\mathbf{A}^j)^T \mathbf{r}^{k,j}$
        $\mathbf{r}^{k,j+1} = \mathbf{r}^{k,j} - \mathbf{A}^j (\mathbf{x}_j^{k+1} - \mathbf{x}_j^k)$
    end for
    $\mathbf{r}^{k+1,1} = \mathbf{r}^{k,q+1}$
end for
```

These two generic algorithms can be applied in parallel by parallelising the inner loop and summing or averaging the updates of $\mathbf{z}^i$ of $\mathbf{r}^{i,j}$ respectively. Each algorithm has their own advantages and drawbacks. For row action method, each sub-iteration does not require the calculation or storage of the entire matrix $\mathbf{A}$ in advance but only needs to calculate a set of rows of $\mathbf{A}_i$ at a time. This can be an advantage in large 3D reconstruction problems where the storage of the whole matrix is infeasible. However, if the algorithm is applied in a parallel form, then each processor (or node) needs to store the whole reconstructed image vector $\mathbf{x}$ as each update is performed on the entire image, which can be computationally challenging in large data reconstructions, especially in terms of the required forward projection $\mathbf{A}_i\mathbf{x}$ and back projection $\mathbf{A}_i^T\mathbf{r}_i$ [11].

On the other hand, using column action algorithms in a parallel computing scheme does not require each processor to store the whole reconstructed image but only to store a small part of $\mathbf{x}$. However, they instead require access to all of $\mathbf{y}$ or $\mathbf{r}$, which again can be prohibitive in large scale situations.

Thus, row action and column action methods require access to the entire vectors $\mathbf{y}$ (or $\mathbf{r}$) or $\mathbf{x}$ in each sub-iteration. There are few exceptions to this. One exception is the work in [26], where a row action method is discussed in which each node only requires parts of the reconstructed image vector $\mathbf{x}$. By splitting the image volume along planes perpendicular to the rotation axis, this variant of SIRT calculates the area of overlap on the detector between two adjacent sub-volumes. Only the data within the overlapping areas on the detector are exchanged between computation nodes. However, this method only works for circular scan trajectories and its scalability is limited due to the requirement that the overlap area should be small to minimise data communication overheads.

Our goal here is to develop algorithms working with generic x-ray tomographic scanning trajectories and with large, realistic data sizes. In this paper, we thus develop a novel algorithm called coordinate-reduced steepest gradient descent (CSGD) that combines aspects of row action and column action methods. Unlike traditional algorithms, our parallelizable algorithm operates on *arbitrary* subsets of both $\mathbf{x}$ and $\mathbf{y}$ at any one time. There are thus no longer any restrictions on scan trajectories or on the way in which the volume is decomposed. The algorithm is thus applicable to arbitrary scan trajectories and is scalable so that it can be run on a range of computing platforms, including low memory GPU clusters and high performance CPU based clusters.

In our algorithm, the reconstruction volume is divided into several sub-volumes, which can be updated separately at different computing nodes. This update is based on a subset of the observations, as well as the reconstructed volume, so that only a small sub-matrix of the system matrix $\mathbf{A}$ is used in each step.

The rest of this paper is organised as follows. The first section gives an overview over our new block iterative method and the second section describes the proposed algorithm in more details. Some simulation results are illustrated in section III and conclusions and further discussions are presented in the last section.

II. Description of CSGD

A. Notation

In this paper we assume a linear, i.e. monochromatic x-ray model [31], [17], [1]:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e}. \tag{2}$$

For now, we use a simple least square cost function to solve this inverse problem:

$$f(\mathbf{x}) = (\mathbf{y} - \mathbf{A}\mathbf{x})^T(\mathbf{y} - \mathbf{A}\mathbf{x}), \tag{3}$$

where $\mathbf{A}$ is the system matrix derived using, for example, Siddon's method [20]. The element A_i^j is the intersection length of the i^{th} x-ray beam with the j^{th} reconstruction voxel. In this paper, I will be an index set that indexes rows in $\mathbf{A}$ (or the subset of x-ray measurements $\mathbf{y}$) $I = i_1, i_2, ...$ Similarly,

J will be a set of column indexes of $\mathbf{A}$ (or the index set of a subset of voxels $\mathbf{x}$) $J = j_1, j_2, ...$ Thus the matrix $\mathbf{A}_{I_i}^{J_j}$ will be a sub-matrix of $\mathbf{A}$ with row indexes I_i and column indexes J_j. Thus we can divide the linear system into several blocks:

$$\begin{bmatrix} \mathbf{y}_{I1} \\ ... \\ \mathbf{y}_{Im} \end{bmatrix} \approx \begin{bmatrix} \mathbf{A}_{I_1}^{J_1} & \mathbf{A}_{I_1}^{J_2} & ... & \mathbf{A}_{I_1}^{J_n} \\ & ... & & \\ \mathbf{A}_{I_m}^{J_1} & \mathbf{A}_{I_m}^{J_2} & ... & \mathbf{A}_{I_m}^{J_n} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{J1} \\ ... \\ \mathbf{x}_{Jn} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{A}_{I_1} \\ ... \\ \mathbf{A}_{I_m} \end{bmatrix} \mathbf{x}. \tag{4}$$

Note that the index sets can be arbitrary partitions of the columns and rows and do not necessarily have to be consecutive. For the convenience of the latter discussion, we also define $\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{x}$ and $\mathbf{r}_I$ as the subset of $\mathbf{r}$ defined as $\mathbf{y}_I - \mathbf{A}_I\mathbf{x}$.

B. Derivation of the algorithm

The key idea in CSGD is to minimize the partial residual $\mathbf{r}_I$ with partial coordinate $\mathbf{x}_J$. In one iteration, after selecting row and column index sets I and J, the object function becomes:

$$f(\mathbf{x}) = \mathbf{r}_I^T\mathbf{r}_I = (\mathbf{y}_I - \mathbf{A}_I\mathbf{x})^T(\mathbf{y}_I - \mathbf{A}_I\mathbf{x}). \tag{5}$$

The steepest descend direction $\mathbf{g}$ of $f(\mathbf{x})$ is then:

$$\mathbf{g} = -\nabla f(x) = \mathbf{A}_I^T(\mathbf{y}_I - \mathbf{A}_I\mathbf{x}) = \mathbf{A}_I^T\mathbf{r}_I. \tag{6}$$

Assume that only those voxels whose indices are in the set J are updated, then the new descend direction becomes:

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_J \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} (\mathbf{A}_I^J)^T\mathbf{r}_I \\ \mathbf{0} \end{bmatrix} \tag{7}$$

and the update on the selected voxels becomes:

$$\mathbf{x}_J^{n+1} = \mathbf{x}_J^n + \mu\mathbf{g}_J, \tag{8}$$

$$\mathbf{x}_{\hat{J}}^{n+1} = \mathbf{x}_{\hat{J}}^n. \tag{9}$$

where μ is the gradient step length and $\hat{J}$ is the complement to the set J.

Ideally, we would like to compute the step length μ such that $f(\mathbf{x}^{n+1})$ is minimised:

$$\nabla f(\mathbf{x}^{n+1})^T\mathbf{g} = 0, \tag{10}$$

where $\nabla f(\mathbf{x}^{n+1}) = (\mathbf{A}_I)^T(\mathbf{A}_I\mathbf{x}^{n+1} - \mathbf{y}_I)$. Using the fact that $\mathbf{A}_I\mathbf{g} = \mathbf{A}_I^J\mathbf{g}_J$, the steepest step length is thus calculated as:

$$\mu = \frac{\mathbf{g}_J{}^T(\mathbf{A}_I^J)^T\mathbf{r}_I}{\mathbf{g}_J{}^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J} = \frac{\mathbf{g}_J{}^T\mathbf{g}_J}{\mathbf{g}_J{}^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J}. \tag{11}$$

With this step length, each iteration only requires the information of $\mathbf{A}_I^J$, which significantly reduces the calculation amount when the matrix is generated on the fly with Siddon's method. Although the calculation of the step size μ still requires access to the error $\mathbf{r}_I = \mathbf{y}_I - \mathbf{A}_I\mathbf{x}$, the calculation of $\mathbf{r}_I$ can be replaced by an update process [30], [27]. Since the update within one iteration only changes $\mathbf{x}_J$, we have:

$$\mathbf{A}_I\mathbf{x}^{n+1} = \mathbf{A}_I\mathbf{x}^n - \mathbf{A}_I^J\mathbf{x}_J^n + \mathbf{A}_I^J\mathbf{x}_J^{n+1}, \tag{12}$$

where $\mathbf{x}_J^n$ is the result of the n^{th} iteration. So the update on $\mathbf{r}_I$ can be written as:

$$\mathbf{r}_I^{n+1} = \mathbf{r}_I^n + \mathbf{A}_I^J(\mathbf{x}_J^n - \mathbf{x}_J^{n+1}) = \mathbf{r}_I^n - \mu\mathbf{A}_I^J(\mathbf{g}_J). \tag{13}$$

We can see that this update only requires computation and storage of $\mathbf{A}_I^J\mathbf{x}_J^{n+1}$.

An important issue is that the step size derived in Eq.11 minimizes $\|\mathbf{r}_I\|$ instead of $\|\mathbf{r}\|$ and that the step length is always positive. However, we are not interested in the reduction of $\mathbf{r}_I$ but in the reduction of $\mathbf{r}$. Our choice of μ can thus potentially be too large to reduce $\mathbf{r}$. Furthermore, it is not guaranteed that our update direction $\mathbf{g}$ is always a descend direction. To stabilise our algorithm, we thus introduce an additional relaxation parameter β into the calculation of μ. This helps us to avoid overshooting the minimum if $\mathbf{g}$ is a descend direction whilst in cases in which $\mathbf{g}$ is not a descend direction, the increase in r remains small. The pseudo-code of the basic computation blocks is shown in Algo.3.

Algorithm 3 The algorithm for a basic iteration

```
Initialization: select system matrix's row index $I$ and column index $J$
$\mathbf{g}_J = (\mathbf{A}_I^J)^T\mathbf{r}_I$
$\mu = \beta\frac{(\mathbf{g}_J)^T\mathbf{g}_J}{(\mathbf{g}_J)^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J}$
$\mathbf{x}_J = \mathbf{x}_J + \mu\mathbf{g}_J$
$\mathbf{z}_I^J = \mathbf{A}_I^J\mathbf{x}_J$
```

It is straightforward to see that these computations can be computed in parallel over J since parameters in the update on $\mathbf{x}_J$ are independent of each other. Further analyses and improvements show that the parallel computation can be even performed over both subsets of J and I.

The main trick in parallelizing the above code is to estimate the error $\mathbf{r}$. Ideally, after the parallel computations have updated subsections of the volume $\mathbf{x}$, we would need to compute a new error vector $\mathbf{r}$, which is required in the computation of subsequent gradient directions. However, this would require the computation of $\mathbf{r}_I^n + \mathbf{A}_I^J(\mathbf{x}_J^n - \mathbf{x}_J^{n+1})$. We instead use a scheme that approximates $\mathbf{r}$. This is done by calculating vectors $\mathbf{z}_I^{J,n+1} = \mathbf{A}_I^J\mathbf{x}_J^{n+1}$.

Algo.4 shows how this is done when parallelizing over blocks J and Algo.5 shows how this can be further parallelized over both blocks J and I. The extension to a parallelization over I is done by averaging over the individual updates of subsets $\mathbf{x}_J$. It is worth noting at this point that these algorithms do not require us to use all subsets J and J in each iteration, but also work if we randomly choose new subsets of these sets in each iteration.

C. Initial numerical examples and comparisons

To show the performance of these methods, we compared convergence results of three algorithms, a standard gradient descend (GD) that uses the full data set, our CSGD method parallelized over J and CSGD parallelized over both I and J. We here generated a matrix $\mathbf{A} \in \mathbb{R}^{200\times 100}$ and the vector $\mathbf{x} \in \mathbb{R}^{100}$ with i.i.d. uniform entries. Our observation was then $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e}$, where $\mathbf{e}$ was a

Algorithm 4 CSGD(J) method

```
Initialization: Partition row and column indices into sets $\{I_i\}$ ($i \in [1, m]$) and $\{J_j\}$ ($j \in [1, n]$).
$\mathbf{x}^0 = \mathbf{0}$, $\mathbf{z}^j_{I_i} = \mathbf{0}$ for all blocks $I_i, J_j$ and $\mathbf{r} = \mathbf{y}$.
for epoch=1,2,... do
  for $k = 1, 2, ..n$ do
    select $J_k$ as index $J$
    for $i$=1,2,..$m$ do
      select $I_i$ as index $I$
      $\mathbf{g}_J = (\mathbf{A}^J_I)^T \mathbf{r}_I$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}^J_I)^T \mathbf{A}^J_I \mathbf{g}_J}$
      $\mathbf{x}_J = \mathbf{x}_J + \mu \mathbf{g}_J$
      $\mathbf{z}^j_I = \mathbf{A}^J_I \mathbf{x}_J$
    end for
    $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$
  end for
end for
```

Algorithm 5 CSGD(I,J) method

```
Initialization: Partition row and column indices into sets $\{I_i\}$ ($i \in [1, m]$) and $\{J_j\}$ ($j \in [1, n]$),
$\mathbf{x}^0 = \mathbf{0}$, $\mathbf{z}^j_{I_i} = \mathbf{0}$ for all blocks $I_i, J_j$ and $\mathbf{r} = \mathbf{y}$.
for epoch=1,2,... do
  $\hat{\mathbf{x}} = \mathbf{0}$
  for $k = 1, 2, ..n$ do
    select $J_k$ as index $J$
    for $i$=1,2,..$m$ do
      select $I_i$ as index $I$
      $\mathbf{g}_J = (\mathbf{A}^J_I)^T \mathbf{r}_I$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}^J_I)^T \mathbf{A}^J_I \mathbf{g}_J}$
      $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu \mathbf{g}_J$
      $\mathbf{z}^j_I = \mathbf{A}^J_I (\mathbf{x}_J + \mu \mathbf{g}_J)$
    end for
    $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$
  end for
  for all blocks $J$ that have been updated, $\mathbf{x}_J = \hat{\mathbf{x}}_J$/(number of times block $J$ has been updated)
end for
```

vector of i.i.d. Gaussian noise with variance of 0.01. In each iteration, we selected 8 subsets of 10 elements J_i and 8 subsets of 20 elements I_i. It should be noted that the GD requires two matrix vector multiplications involving the matrix $\mathbf{A}$ per iteration whilst CSGD requires 3 multiplications with each of the sub-matrices. As matrix vector multiplications are the dominant factor, the above number of subsets guaranteed roughly equal numbers of computations per iteration for each algorithm.

Typical convergence results (after some tuning of parameter β, more on this later) are shown in figure 1. Here we see that by parallelizing over J the CSGD even outperforms the gradient descend method, something that was observed frequently. Also evident is the fact that parallelizing over J only tends to work better in the long run, though in the early iterations, it is found that the fully parallel version tends to perform best.

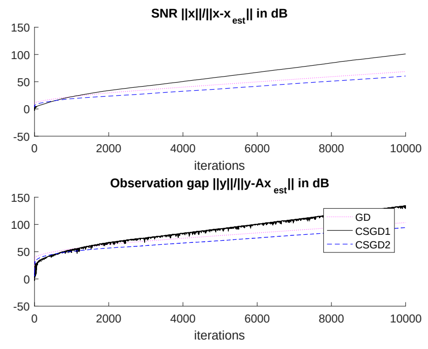


Fig. 1: Comparisons of gradient descend (GD, dotted) and two parallel versions of CSGD, that are parallelized over J only (CSGD1, solid) and over both I and J (CSGD2, dashed).

D. The importance sampling strategy

Based on the proposed algorithms, it is straightforward to develop a random sampling strategy that goes through all projection views and all detector sub-areas arbitrarily. Looking at algorithms 4 and 5, we can either split the problem by selecting all subsets $\{I_i\}$ and $\{J_j\}$ in each outer iteration. Alternatively, we could randomly select subsets of $\{I_i\}$ and $\{J_j\}$, with each set being chosen with equal probability. Considering the sparsity of $\mathbf{A}$ and taking inspiration from the randomized Kaczmarz method in [32], we instead develop an importance sampling strategy. Sets in $\{I_i\}$ and $\{J_j\}$ are selected with a probability that is proportional to the sparsity of the sub-matrix $\mathbf{A}_I^J$. To estimate this sparsity without the need to construct the full matrix $\mathbf{A}$, we instead compute the overlap between the detector area and the projection of the voxels labelled by J. Using the importance sampling strategy for I and iterating over partial J, Algo.4 and 5 become Algo.6 and 7:

It should be noted that although in Algo.6 and 7 the update on $\mathbf{r}$ is performed as a whole, in effect, only elements which have been chosen for the last time iteration have been changed. As a result, we can also take the union of all selected Is and call this set It. The update of $\mathbf{r}$ then becomes $\mathbf{r}_{It} = \mathbf{y}_{It} - \sum_j \mathbf{z}_{It}^j$.

Algorithm 6 CSGD(J) method with importance sampling

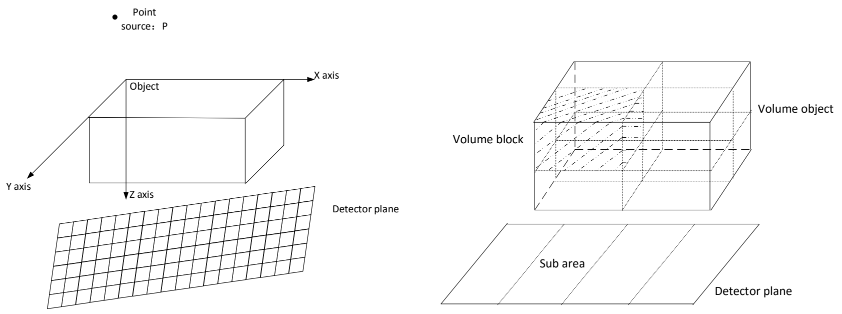
```
Initialization: Partition row and column indices into sets $\{I_i\}$ ($i \in [1, m]$) and $\{J_j\}$ ($j \in [1, n]$), $\mathbf{x}^0 = \mathbf{0}$. $\mathbf{z}_I^j = \mathbf{0}$ for all row blocks $I_i$ and $J_j$, $\mathbf{r} = \mathbf{y}$. $\gamma$ is the percentage of selected volume blocks in the total volume blocks. $\alpha$ is the percentage of selected row blocks in the total row blocks. Relaxation parameter $\beta$ is defined as $b * \frac{P_S}{P_T}$, where $P_s$ is the projection area of the selected sub volume on the selected sub area on current iteration and $P_T$ is the total projection area for the selected volume object on the total detector plane under the whole scanning trajectory.
For each pair of indices from $\{I_i\}$ and $\{J_j\}$ compute the overlap $P(I, J)$ between the projection of the volume $\mathbf{x}_J$ and the subset of all detector pixels indexed by $I$.
for epoch=1,2,... do
  for  $k = 1, 2, ..\gamma n$ do
    select a column block $J$ from $\{J_j\}$ randomly
    for $l = 1, 2, ...\alpha m$ do
      select a row block $I$ from $\{I_i\}$ without replacement with probability proportional to $P(I, J)$
      $\mathbf{g}_J = (\mathbf{A}_I^J)^T \mathbf{r}_I$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}_I^J)^T \mathbf{A}_I^J \mathbf{g}_J}$
      $\mathbf{x}_J = \mathbf{x}_J + \mu \mathbf{g}_J$
      $\mathbf{z}_I^j = \mathbf{A}_I^J \mathbf{x}_J$
    end for
    $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$
  end for
end for
```

E. Domain decompositions in tomography.

The above algorithms have been designed so that each computation is carried out on a generic subset I (i.e. a subset of the observations) and a generic subset J (i.e. a subset of the voxels) at any one time. For tomographic reconstruction, the question thus arises how to partition the observations and the reconstruction volume. Whilst generic partitions are possible, given the need to compute $P(I, J)$, it makes sense to partition the reconstruction volume and detector areas into blocks. Let us use a 3D cone beam CT geometry (similar arguments can be made for a parallel beam setup). The reconstruction volume and one pair of source/detector locations are shown in Fig.2(a). For simplicity, the detector plane is always perpendicular to the line connecting point source and the geometry center of the detector plane. We label each location of the point source S and detector location with parameter θ, whose trajectories do not have to be circular or helical. For each source/detector location θ, we partition the detector into blocks. We also partition the reconstruction volume into rectangular cuboids, as shown in Fig.2(b). For such a partition, given any one block of the reconstruction volume J, for each source location θ, there might only be a part of the detector area that is involved in the update

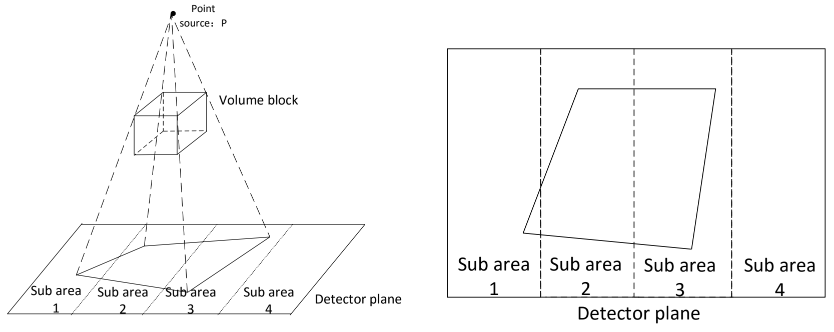
Algorithm 7 CSGD(I,J) method with importance sampling

```
Initialization: Partition row and column indices into sets $\{I_i\}$ ($i \in [1,m]$) and $\{J_j\}$ ($j \in [1,n]$), $\mathbf{x}^0 = \mathbf{0}$. $\mathbf{z}_I^j = \mathbf{0}$ for all row blocks $I_i$ and $J_j$, $\mathbf{r} = \mathbf{y}$. $\gamma$ is the percentage of selected volume blocks in the total volume blocks. $\alpha$ is the percentage of selected row blocks in the total row blocks. Relaxation parameter $\beta$ is defined as $b * \frac{P_S}{P_T}$, where $P_S$ is the projection area of the selected sub volume on the selected sub area on current iteration and $P_T$ is the total projection area for the selected volume object on the total detector plane under the whole scanning trajectory.
For each pair of indices from $\{I_i\}$ and $\{J_j\}$ compute the overlap $P(I,J)$ between the projection of the volume $\mathbf{x}_J$ and the subset of all detector pixels indexed by $I$.
for epoch=1,2,... do
    $\hat{\mathbf{x}} = \mathbf{0}$
    for $k = 1,2,..\gamma n$ do
        select a column block $J$ from $\{J_j\}$ randomly
        for $l = 1,2,...\alpha m$ do
            select a row block $I$ from $\{I_i\}$ without replacement with probability proportional to $P(I,J)$
            $\mathbf{g}_J = (\mathbf{A}_I^J)^T \mathbf{r}_I$
            $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}_I^J)^T \mathbf{A}_I^J \mathbf{g}_J}$
            $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu \mathbf{g}_J$
            $\mathbf{z}_I^j = \mathbf{A}_I^J (\mathbf{x}_J + \mu \mathbf{g}_J)$
        end for
        $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$
    end for
    for all blocks $J$ that have been updated, $\mathbf{x}_J = \hat{\mathbf{x}}_J$/(number of times block $J$ has been updated)
end for
```



(a) (b)

Fig. 2: Geometry of a 3D scanning model and a partition method on both reconstruction volume and detector area.



(a) (b)

Fig. 3: (a) shows a 3D model of the cone beam setup with one block of the volume being projected on a detector plane. (b) shows the projection area of the volume block, whose projection is unevenly distributed on each of sub-area.

in CSGD. Thus, for a given J, the sub matrices $\mathbf{A}_I^J$ can have different levels of sparsity for different I. An illustration of this is shown in Fig.3, which shows that in this fixed view, sub-area 4 on the detector does not receive rays passing through the selected sub volume and so there is no need to select this area to update $\mathbf{x}_J$. What is more, the bulk of the volume block projection mainly lies in sub-areas 2 and 3 of the detector so that the corresponding $\mathbf{A}_{Isub2}^J$ and $\mathbf{A}_{Isub3}^J$ are much more dense than $\mathbf{A}_{Isub1}^J$. This suggests that the algorithm should select sub-area 2 and 3 more frequently than sub-area 1 and this is here achieved with our importance sampling strategy.

The above derivation is illustrative. In a real setup, where we have hundreds if not thousands of source/detector locations, it might at a first glance make little sense to partition the observations $\mathbf{y}$ into the small sets implied by a partition of the detector. Nevertheless, it might be of interest to partition $\mathbf{y}$ into larger groups of these subsets. That is, each partition contains subsets of the detector for a large number of different projections. For example, if we have a traditional, rotation based scan trajectory in 3D, then it would make sense to partition the detector as well as the reconstruction volume in the direction parallel to the rotation axis, as this would lead to sub-matrices $\mathbf{A}_I^J$ that are either very sparse or very dense. What is more, it would even be possible to dynamically create subsets of $\mathbf{y}$ that differ between epochs. We call such an approach group CSGD (see below).

To demonstrate the advantage of importance sampling based on the sparsity of $\mathbf{A}_I^J$, we studied the convergence on a problem similar to the one above. The difference here was that we partitioned the matrix $\mathbf{A} \in \mathbb{R}^{200\times 100}$ into 4 blocks of size 100×50, with one block being dense (i.e. having 5000 non-zero entries), two blocks having only 500 non-zero elements and one block having 1667 non-zero elements. Non-zero elements were drawn from the uniform distribution and so were elements in $\mathbf{x} \in \mathbb{R}^{100}$. Our observation was then $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{e}$, where $\mathbf{e}$ was a vector of i.i.d. Gaussian noise with variance of 0.01. In each iteration, we again selected 8 subsets of 10 elements J_i and 8 subsets of 20 elements I_i.

The main difference to the above simulations were that for different I and J, we chose β depending on properties of the matrix $\mathbf{A}_I^J$. For the importance sampling based algorithm, we chose β as the percentage of non-zero rows over the total rows of $\mathbf{A}_I^J$. For the simulation without importance sampling, we further multiplied each β value by $P(I,J)$, where $P(I,J)$ is the fraction of non-zero entries in the matrix $\mathbf{A}_I^J$, e.g. P is 1 if the matrix is dense and 0 if it has only zero entries. Otherwise the algorithm without importance sampling strategy did not converge.

Both algorithms used the version that is parallelized over both I and J. For the importance sampling version of the algorithm, within each epoch, we uniformly selected 8 out of the 10 elements in $\{I_i\}$. For each I, we then used the probability $p(j) = P(I, J_j)/\sum_j(P(I, J_j))$ to sample 8 elements from $\{J_j\}$ without replacement. Typical convergence results (averaged over 10 simulations) are shown in figure 4. It is clear that importance sampling increases convergence. We here sampled I uniformly and then, based on I, we sampled J using the probability derived from $P(I,J)$. The alternative to sample from J uniformly and from I depending on $P(I,J)$ also worked better than sampling uniformly from both I and J.

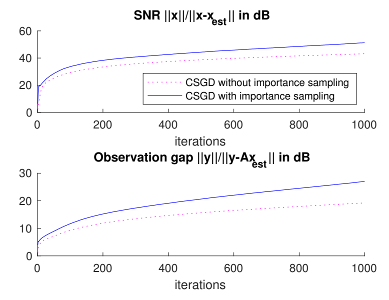


Fig. 4: Comparisons of CSGD with (solid) and without (dotted) importance sampling

F. Group CSGD

The advantage of having dense sub-matrices leads to the following group version of our algorithms. The idea here is to dynamically build larger, dense sub-matrices $\mathbf{A}_I^J$ out of a large selection of smaller sub-matrices. In the previous CSGD method, one sub-block of the image combines only one sub-detector area for one projection view. In GCSGD, the sub-block combines a group of sub areas for several projection views. It is straightforward that the GCSGD method uses more row information than the CSGD method. Let us demonstrate the idea by modifying algorithm 7.

The difference here is that we partition row indices into *many small* sets $\{I_i\}$ whilst we partition the column indices into a *few larger* sets $\{J_j\}$. Within each epoch, we then group the smaller sets $\{I_i\}$ into few larger sets $\{I_g\}$, which are then used for the iteration of the method.

G. Computational complexity

There are several important aspects when comparing computational efficiency of the methods. The methods are designed to allow parallel computation. We envisage this to be performed in a distributed

Algorithm 8 GCSGD(I,J) method with importance sampling

```
Initialization: Partition row and column indices into sets $\{I_i\}(i \in [1, m])$ and $\{J_j\}(j \in [1, n])$, $\mathbf{x}^0 = \mathbf{0}$. $\mathbf{z}_I^j = \mathbf{0}$ for all row blocks $I_i$ and $J_j$, $\mathbf{r} = \mathbf{y}$. $\gamma$ is the percentage of selected volume blocks in the total volume blocks. $\alpha$ is the percentage of selected row blocks in the total row blocks. Relaxation parameter $\beta$ is defined as $b * \frac{P_S}{P_T}$, where $P_S$ is the projection area of the selected sub volume on the selected sub area on current iteration and $P_T$ is the total projection area for the selected volume object on the total detector plane under the whole scanning trajectory. Group size is set as $s$.
For each pair of indices from $I_i$ and $J_j$ compute the overlap $P(I, J)$ between the projection of the volume $\mathbf{x}_J$ and the subset of all detector pixels indexed by $I$.
**for** epoch=1,2,... **do**
  $\hat{\mathbf{x}} = \mathbf{0}$
  **for** $k = 1, 2, ..\gamma n$ **do**
    select a column block $J$ from $\{J_j\}$ randomly
    $l = 0$
    **while** $l < \alpha m$ **do**
      $I_g = \mathbf{0}$
      $pin = 0$
      **while** $pin < s$ **do**
        $pin = pin + 1$
        $l = l + 1$
        select a row block $I$ from $\{I_i\}$ without replacement with probability proportional to $P(I, J)$
        $I_g = I_g \cup I$
      **end while**
      $\mathbf{g}_J = (\mathbf{A}_{I_g}^J)^T \mathbf{r}_{I_g}$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}_{I_g}^J)^T \mathbf{A}_{I_g}^J \mathbf{g}_J}$
      $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu \mathbf{g}_J$
      $\mathbf{z}_{I_g}^j = \mathbf{A}_{I_g}^J (\mathbf{x}_J + \mu \mathbf{g}_J)$
    **end while**
    $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$
  **end for**
  for all blocks $J$ that have been updated, $\mathbf{x}_J = \hat{\mathbf{x}}_J$/(number of times block $J$ has been updated)
**end for**
```

network of computing nodes[1]. Most of these nodes will be used to perform the parallel computations. They produce two outputs,

1) $\hat{\mathbf{x}}_J(i) = \mathbf{x}_J^n + \mu \mathbf{g}_J$
2) $\mathbf{z}_I^j = \mathbf{A}_I^J \mathbf{x}_J$.

These are then either sent to larger, but slow storage or directly to other nodes, where they are eventually used to compute

1) $\mathbf{x}_J = \text{mean}_i(\hat{\mathbf{x}}_J(i))$
2) $\mathbf{r} = \mathbf{y} - \sum_j \mathbf{z}^j$ or $\mathbf{r}_{It} = \mathbf{y}_{It} - \sum_j \mathbf{z}_{It}^j$,

which can be performed efficiently using message passing interface reduction methods.

The three main points that affect performance of the method are:

1) Computational complexity in terms of multiply add operations.
2) Data transfer requirements between data storage and a processing unit as well as between different processing units.
3) Data storage requirements, both in terms of fast access RAM and in slower access (e.g. disk based) data storage.

Each of these costs are dominated by different aspects:

1) Computational complexity is dominated by the computation of matrix vector products involving $\mathbf{A}_I^J$ and its transpose, especially as $\mathbf{A}$ is not generally stored but might have to be re-computed every time it is needed. The computational complexity is thus $O(|I| * |J|)$, though computations performed on highly parallel architectures, such as modern GPUs, are able to perform millions of these computations in parallel.
2) Data transfer requirements are dominated by the need for each of the parallel computing nodes to need $\mathbf{r}_I$ and $\mathbf{x}_J$ as input and $\hat{\mathbf{x}}_J(i)$ and $\mathbf{z}_I^j$ as output. Note that the size of the required input and output vectors are the same, the data transfer requirement is thus $O(|I| + |J|)$.
3) Central data storage requirements are dominated by the need to store the original data and the current estimate of $\mathbf{x}$. We also need to compute and store averages over $\hat{\mathbf{x}}_J(i)$ and $\mathbf{z}_I^j$. These computations can be performed efficiently using parallel data reduction techniques. Our approach would mean that each node would thus require $O(|I| + |J|)$ local memory.

III. Simulations

Before introducing the simulation results, let us introduce three important parameters:

α: is the percentage of selected row blocks (detector areas) for one column block (volume block).

β: is the relaxation parameter tuning the update on x_J. It is expressed as $\beta = b * \frac{P_S}{P_T}$, where P_S is the projection area of the selected sub-volume on the selected sub-area and P_T is the total projection

[1]A serial version running on a single computing node where each computation is done independently, but one after the other, is also possible and this is how many of the simulations reported here were computed.

area for the selected volume object on the total detector plane under the whole scanning trajectory. In the following simulations we mainly change the value of b to tune the relaxation parameter β.

γ: is the percentage of the selected column blocks (volume blocks) during one iteration. In our simulations, γ is usually set as 1, i.e. all column blocks are selected during one iteration.

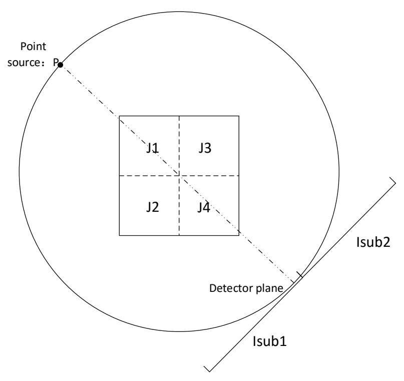
We explored the performance of Algo.8 on a range of tomographic reconstruction problems. We started with a simulated 2D phantom with 64*64 pixels. Pixel sizes were normailsed to be 1. The point source adopted a circular trajectory with radius of 115 and the rotation centre was located at the centre of the object. The linear detector had 187 pixels whose spacing is 1 and the detector was always perpendicular to the line connecting the point source and the geometric centre of the linear detector. The detector centre also followed a circular trajectory with the same radius as the source. The number of projections was 360 with the angular intervals being 1°. The object was partitioned into 4 parts (2 parts in both vertical and horizontal directions) and the detector area was partitioned into 2 parts by default. Scan geometry including the partition method as well as the original phantom are shown in Fig.5. We used the parallel computing toolbox (version 6.8) in Matlab R2016a to perform CSGD as described in Algo.7 and GCSGD in Algo.8. In our simulation, the term *epoch* refers to the outer iteration. The number of sub-matrices $\mathbf{A}_I^J$ that the algorithm used per epoch was proportional to the parameter α, where $\alpha = 1$ means that we used all sub-matrices of $\mathbf{A}$, whilst $\alpha = 0.5$ means that only half of the sub-matrices were used. As the computational load is dominated by matrix vector products, when we compare the difference in convergence rate for methods using different αs (for example, $\alpha = 1$ and $\alpha = 0.5$), we take account of the reduction in computational effort when using smaller α by scaling the epoch count by multiplying it by α, a measure we call *effective epoch*.

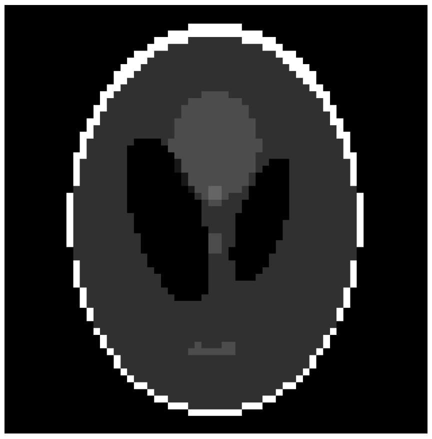
As mentioned before, CSGD and GCSGD methods can be applied in both using importance sampling and random sampling strategies.

We run each algorithms for over one thousand iterations and show their average convergence and stability. However, in realistic applications we are typically only interested in the initial iterations. For example, the convergence performance within 50 effective epochs. This is because in real applications, due to the large data sets that have to be processed and due to the influence of noise and model missmatch, performing more iterations is typically not feasible and is often not helpful in obtaining better reconstruction results.

A. Influence of β

We start with an evaluation of the optimal choice of β, which is crucial for the performance of the method. A simulation was conducted to show the difference in convergence rate when changing b. Results are shown in Fig.6.





(a) (b)

Fig. 5: Basic situation settings. (a) shows the scanning geometry and partition model in the 2D simulation. The object is partitioned from J_1 to J_4 and the detector is partitioned to two parts $Isub_1$ and $Isub_2$. The total 187 detector pixels are assigned to these two sub areas without overlapping. (b) shows the original image to be scanned and reconstructed.

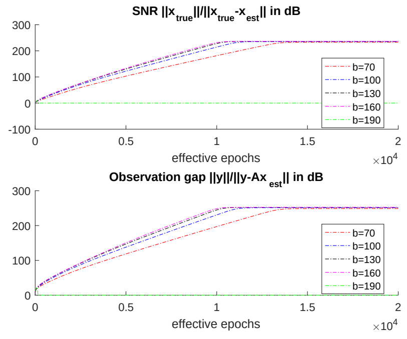


Fig. 6: Different β lead to different convergence rates when reconstructing the 2D image. α and group size are both set as 1. All 4 volume sub-blocks are selected in each epoch.

It can be seen that from $b = 70$ to $b = 160$, larger β increases the convergence rate. However, it was found that further increasing the value of b to 190 leads to a divergence of the algorithm, which suggests that there is a range for β that is guranteed to converge. However, currently the justification for our choice of β remains empirical and more analysis of the algorithm is required to fully understand the optimal parameter.

B. *Choice of* α.

When we set $\alpha = 1$, the importance sampling strategy cannot show any advantage since we always use all data. To demonstrate the difference, we set the group size to 1, $b = 100$ and set α to 0.8,0.5 and 0.2 respectively. Fig.7 shows that setting α smaller than 1 increases the convergence rate of the CSGD. Furthermore, different values of α always lead to the same precision.

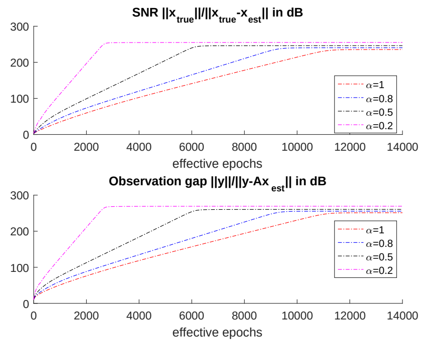


Fig. 7: When $b = 100$, groups size is 1 and all 4 volume blocks are selected for each epoch, setting α smaller than 1 is guaranteed to achieve high precision level and is of help to increase the convergence rate.

Considering that the smaller α is, the more actual epochs are required to achieve the same number of effective epochs and that MATLAB is not efficient in performing loops, in the following simulations, we chose a moderate parameter $\alpha = 0.5$ to test other qualities of CSGD and GCSGD. For example, we test the influence of group size of the GCSGD. Simulation results are shown in Fig.8.

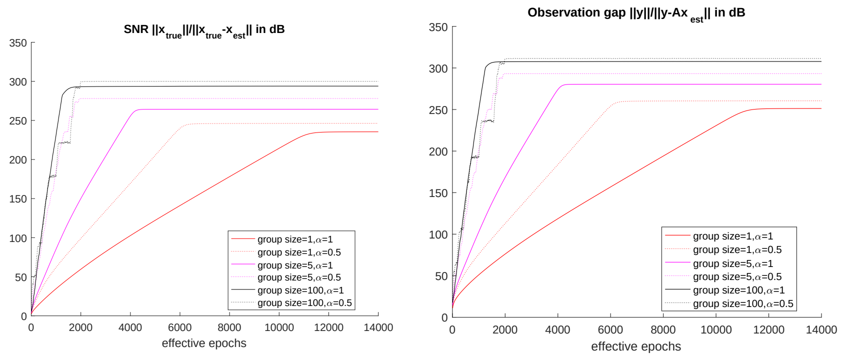


Fig. 8: Different group sizes lead to different precision levels and different convergence rates. The parameter b is tuned to ensure GCSGD with different group sizes does converge. From group size 1 to group size 100, the corresponding b is 100,25,2 respectively. Some *flat areas* in group size=100 appear and the reason will be explained later.

It can be seen that the GCSGD with importance sampling strategy all converges to a high precision level regard less of the group size. Generally speaking, setting the group size as large as possible helps to increase the initial convergence rate. This is reasonable since more row information is used for each epoch when the group size is enlarged, making the iteration more likely to move towards the global *downhill* direction. Furthermore we empirically observed that a larger group size requires smaller parameter b to maintain convergence. This is because when the group size increases, the ratio $\frac{P_S}{P_T}$ also increases and thus a smaller b is required, otherwise it is easy for the step to overshoot, which makes the algorithm fail to converge. In practice, however, for parallel computing architectures, we suggest that the group size should be designed depending on the storage ability of each computing node. To show differences when using different group sizes and different αs, some reconstruction results are presented in Fig.9.

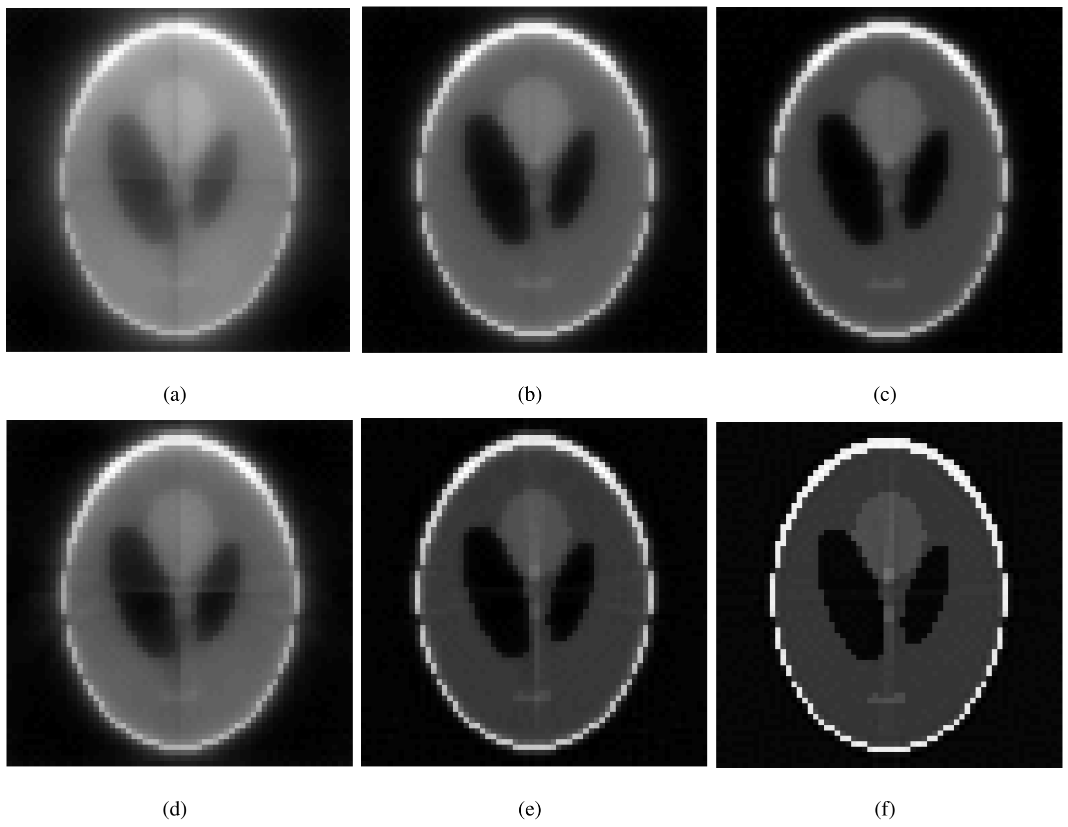


Fig. 9: Simulation results after 20 effective epochs.(a) group size=1, $\alpha = 1$, $b = 100$, SNR=3.44dB; (b) group size=5, $\alpha = 1$, $b = 25$, SNR=6.03dB; (c) group size=100, $\alpha = 1$, $b = 2$, SNR=7.75dB; (d) group size=1, $\alpha = 0.5$, $b = 100$, SNR=5.43dB (e) group size=5, $\alpha = 0.5$, $b = 25$, SNR=11.42dB (f) group size=100, $\alpha = 0.5$, $b = 2$, SNR=23.76dB. When α is 0.5, the reconstructed images are blurred by artefacts. The reasons and the solutions will be discussed later.

C. Different partitions

The above simulations show the effectiveness of the CSGD and GCSGD method with importance sampling. In the following simulation, we explored the influence of different partition methods. For example, the reconstructed image was still divided into 4 square blocks and the detector area was partitioned into 2,4,8 and 16 sets respectively. We compared the difference between group size being 1 and 100 to further verify that a larger group helps to increase the convergence rate. The simulation results are shown in Fig.10.

When the group size is only one, we can see that dividing the detector into 8 and 16 areas leads to rather slow convergence rates. It suggests that when the group size is small, the detector cannot be divided into more sub-areas. For a group size of 100, however, the algorithm allows for a devision

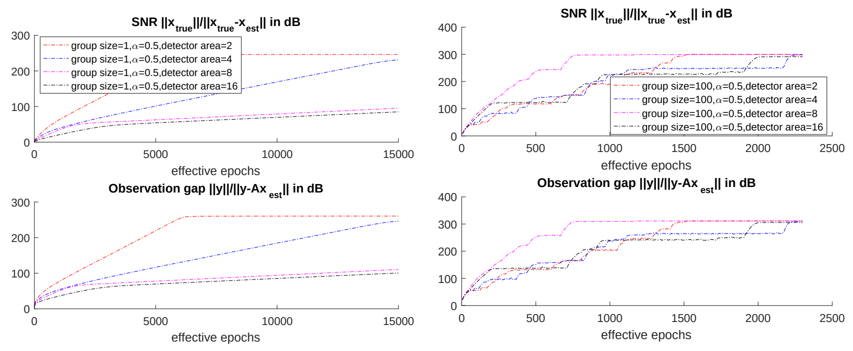


Fig. 10: When the group size is 1 and 100 the convergence results for different detector partition numbers. For group size is 1, the b is set as 100,200,400,800 respectively for detector areas being 2 to 16. For the group size is 100 situation, the b is 2,4,8,16 respectively

of the detector into more areas and the final precision level is not affected. In the initial iterations, we can see that different detector partitions have nearly the same convergence rate in the GCSGD method. When the precision level is high, the *flat areas* appear again and the reason will be explained later. In fact, the *flat area* is of less interest to use since it only appears when the reconstructed image already has a very high precision level and never appears for the initial iteration stage. As a result, this simulation suggests that within a range, reducing the row information (partition the detector area from 2 to 4 or even bigger when group size is fixed) does not influence the initial convergence rate when using GCSGD, especially when the group size is large. It is straightforward to see that the calculation amount of A_I^J for a selected sub-volume and sub-detector area decreases when the number of partitions on the detector increases. As a result, when addressing the partition strategy for large scale tomography problems, a more flexible partition method is allowed for large group size situation. If the big group sizes poses a heavy storage burden for work nodes, a finer partition method would be of help to reduce the computation amount for $\mathbf{A}_I^J$. In other words, although dividing the detector area into more sub-areas cannot provide a higher precision level, it does help to reduce the computation load when only a few iterations are performed.

Another variable is the number of volume blocks. To investigate this, we partitioned the 2D image into 4,16,64,256 areas respectively (divide each dimension into 2,4,8,16 parts evenly). The detector area was always partitioned into 2 sub areas, as shown in Fig.5. The group size was set to 100 and the corresponding b was 2,1,0.5,0.25 respectively to ensure convergence. The simulation results are shown in Fig.11. It can be seen that all partition methods show similar final precision level, and in the initial iterations, the differences between different volume block numbers are also negligible, demonstrating the good scalability of this algorithm. Since a finer partition allows more computation nodes to be used and the computation load of each node is reduced, the similar convergence in the initial iterations is useful when addressing large scale data where only a few iterations are performed.

We also explored the situation when not all of J were selected. The group size was set to 100

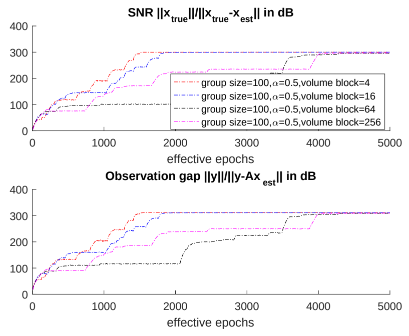
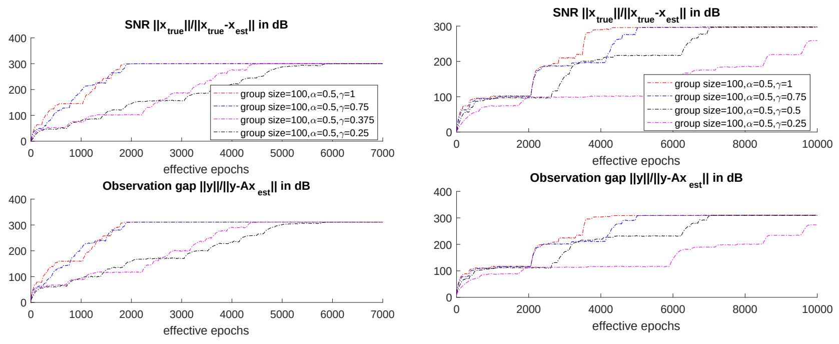


Fig. 11: When the image is divided into different volume slices, and all volume blocks are selected for each epoch, the final precision level and the initial convergence rate is similar.



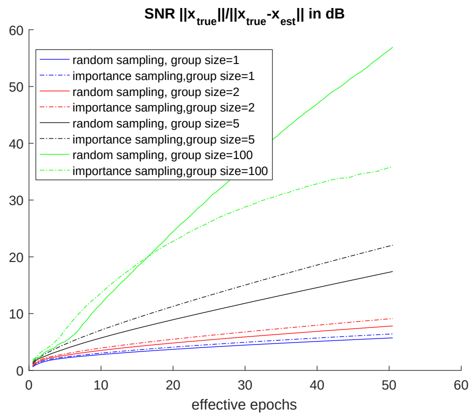
(a) When the image is partitioned into 16 square blocks, $b = 1$ for all simulations

(b) When the image is partitioned into 64 square blocks,$b = 0.5$ for all simulations

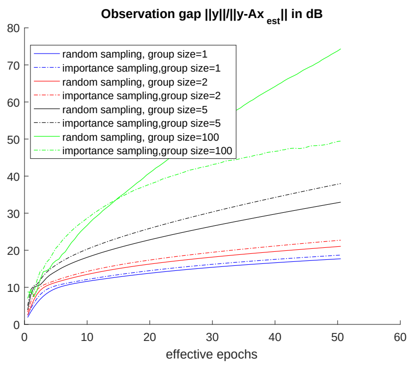
Fig. 12: When randomly select $\gamma * 100\%$ image blocks to get involved in iteration for each epoch. Despite of the flat areas, the final precision levels between different γ are the same. The initial convergence rates are similar when the γs are large than 0.5. When γ is 0.25, the convergence rate slows down.

and the number of detector sub-areas was set to 2 by default. We defined γ as the percentage of the number of selected image blocks over the total number of image blocks and the γ was set as different values when the image were partitioned into 16 and 64 sub areas, simulation results are shown in Fig.12.

It can be seen that GCSGD method converges for every γ, and their final precision level is similar. Also, as long as the γ is not too small ($\gamma = 0.2$), then the initial convergence rates for different γ are almost identical. This suggests that randomly leaving a small portion of the image blocks not to be updated does not influence the initial convergence rate and final precision level. This property will be useful when we perform the algorithm in parallel computing architecture where the algorithm maybe run asynchronously.



(a)



(b)

Fig. 13: The importance sampling strategy always outperforms random sampling when group size is 1, 2 and 5. When the group size is 100, the importance sampling strategy only works at the early stage, and then the advantages over random sampling disappears.

D. *Discussions on importance sampling strategy*

In this part we compared the importance sampling and random sampling strategy based on the previous simulation geometry. Since the convergence property has been verified in previous simulations, in this part, we were only interested in the initial convergence rate and thus we only perform the algorithm (with importance sampling or random sampling strategy) for 20 effective epochs but repeat it 10 times for the same data and then calculate the average of SNR and observation GAP.

The simulations not only show the drawbacks and advantages of the importance sampling strategy, but also explain why the *flat area* appears when the precision level is high and group size is large, and why using importance sampling can lead to images artefacts.

We first verified that the importance sampling strategy indeed helps to increase the convergence rate compared with random sampling. However, it only happens when the group size is small. The simulation is shown in the Fig.13. The reason is due to the non-uniform update on $\mathbf{z}_I^j$ when we use the importance sampling strategy. For each volume slice, the selection criteria for choosing row blocks (or sub-detector areas from different projection views) is based on the projection area, which reflect the sub matrix's density. The difference between projection areas can sometimes be huge. For example, as shown in Fig.14, for volume slice J_1, the projection area on the sub-detector 2 is far larger than that on the sub-detector 1. That means when updating J_1 and selecting sub-detector areas, or row blocks, it is more likely to that sub-detector 2 is chosen. This property makes the update of $\mathbf{z}_{I_2}^{J_1} = \mathbf{A}_{I_2}^{J_1}\mathbf{x}_{J_1}$ very frequent and the update on $\mathbf{z}_{I_1}^{J_1} = \mathbf{A}_{I_1}^{J_1}\mathbf{x}_{J_1}$ rather infrequent. For residue terms $\mathbf{r}_{I_1}$ and $\mathbf{r}_{I_2}$, the contribution from $\mathbf{z}_{I_1}^{J_1}$ and $\mathbf{z}_{I_2}^{J_1}$ is different, especially in the initial iterations. Since the density of $\mathbf{A}_{I_2}^{J_1}$ is higher than $\mathbf{A}_{I_1}^{J_1}$, this means the influence of $\mathbf{z}_{I_2}^{J_1}$ on $\mathbf{r}_{I_2}$ is more significant than $\mathbf{z}_{I_1}^{J_1}$ on $\mathbf{r}_{I_1}$. At the initial iterations, the frequent updates of $\mathbf{z}_{I_2}^{J_1}$ make r_{I_2} decrease efficiently while the infrequent

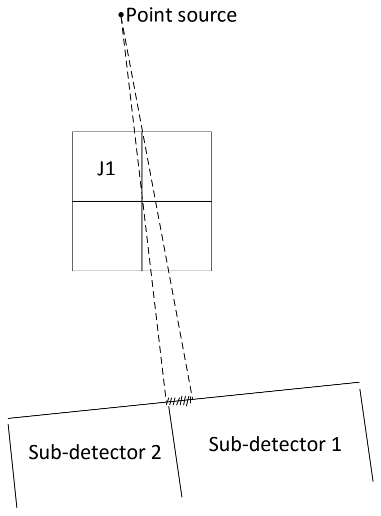


Fig. 14: The difference of the projection area for one projection view can be huge for some locations. Here I_1 means the row blocks for sub-detector 1 in the current projection view and I_2 has similar definitions for sub-detector 2.

updated of $\mathbf{z}_{I_1}^{j_1}$ has little influence on the update of $\mathbf{r}_{I_1}$. (It should be remembered that at the same time $\mathbf{r}_{I_1}$ is also updated by computations with the other volume slices.) This property helps the importance sampling strategy to obtain a fast initial convergence rate but when the precision level is already high, the update on $\mathbf{r}_{I_1}$ then needs the contribution from $\mathbf{z}_{I_1}^{J_1}$ and unless this term is updated frequently, then $\mathbf{r}_{I_1}$ cannot be further reduced. This explains why at the initial stage the importance sampling strategy is faster than random sampling and when the iteration goes on, there are some *flat areas* in Fig.10, Fig.11, Fig.12. The reason for the artefacts in the reconstructed images is similar: the shaded area in Fig.14 is rarely used to update J_1, thus allowing the reconstruction error of the inner margin of J_1,(the same hold in the other volume slices) a little higher than IN other parts. In terms of why the importance sampling strategy works and the *flat areas* disappears when the group size is small (for example, group size is set as 1) is because that IN this situation, the total convergence rate is slow and when the precision level reaches a high level, then the update on $\mathbf{z}_{I_1}^{J_1}$ is frequent enough to allow an effective change on r_{I_1}.

We proposed some methods to overcome the disadvantages of importance sampling. One method would be to allow the algorithm to make a choice before sampling the row blocks, deciding whether to use importance or random sampling in the current iteration. This choice can be done with a specific probability. Another method would define a threshold in the probability distribution, enforcing it to be above the threshold. However, simulation results show that these two methods are not very effective in terms of improving the performance of importance sampling strategy. The third method, we call a mixed sampling strategy, is to gradually change the probability distribution along with the iterations. By reducing the gap between different probability values gradually, we experimentally showed that this

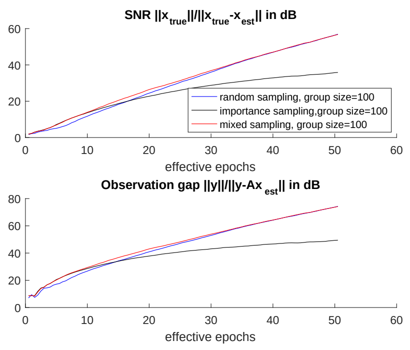


Fig. 15: Comparisons for different sampling strategies. The θ in mixed sampling method starts from 0 and the increment step length is 1/40, it means that after 40 epochs the probability distributions on different sub-detector areas becomes the same within one projection.

method can maintain the fast initial convergence due to the importance sampling while eliminating the artefacts in the reconstructed image and the *flat area* in the convergence line. Specifically, in Fig.14, we defined the projection area on sub-detector area 1 as P_1 and that on sub-detector area 2 as P_2. In the previous importance sampling strategy, the probability for choosing sub-detector area 1 is $\frac{P_1}{P_T}$, where P_T is the total projection area of sub-volume J_1. Similarly, the probability for choosing sub-detector 2 is $\frac{P_2}{P_T}$. The mixed sampling method fills the gap between $\frac{P_1}{P_T}$ and $\frac{P_2}{P_T}$. The probabilities to choose sub-detector 1 and 2 are set proportional to $\frac{P_1+\theta(P_{max}-P_1)}{P_T}$ and $\frac{P_2+\theta(P_{max}-P_2)}{P_T}$ respectively, where P_{max} is the largest value of P_1 and P_2. The parameter θ is used to reduce the difference between two possibilities by gradually increasing it from 0 to 1. We make a comparison with importance sampling, random sampling and the mixed sampling method when the group size is 100 and $\alpha = 0.5$. The convergence rate is shown in Fig.15. We can see that the mixed sampling method maintains the same convergence rate at the initial stage as importance sampling strategy does, while keeping this increment trend along with the iteration going on.

We also present the reconstructed image after 20 effective epochs for group size being 1,5,and 100 respectively. The simulation condition is the same as Fig.9. For simplicity, we only simulated the $\alpha = 0.5$ situations. The simulation results are shown in Fig.16.

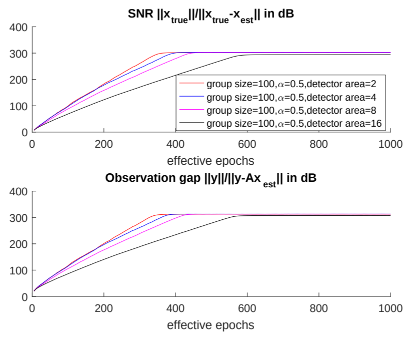
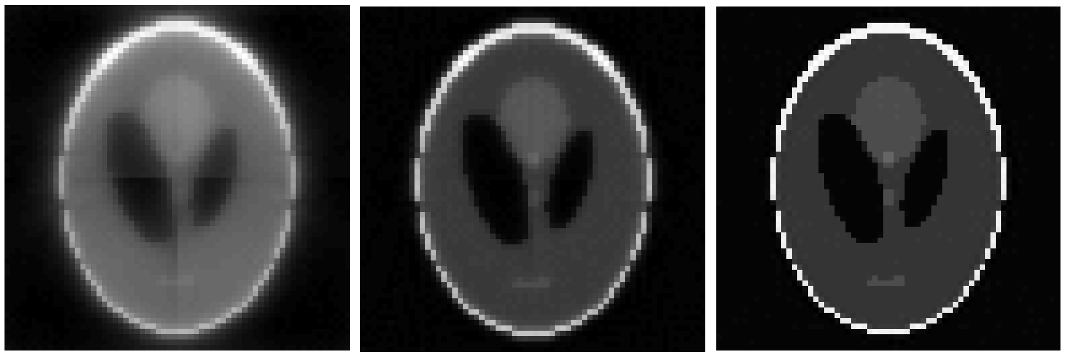


Fig. 17: Using the mix sampling strategy eliminate the *flat area*, thus shorten the iteration numbers needed to achieve the precision limit.



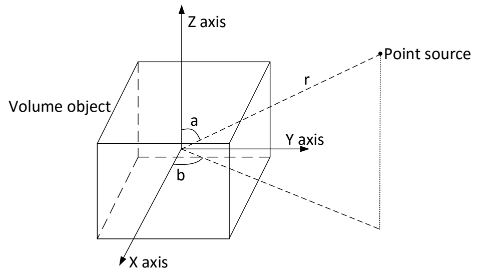
(a) (b) (c)

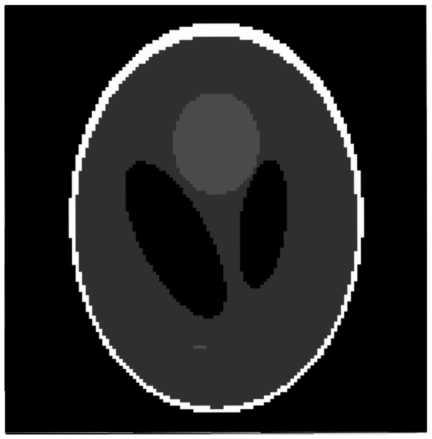
Fig. 16: Simulation results after 20 effective epochs when use mixed sampling strategy.(a) group size=1, $\alpha = 0.5$, $b = 100$, SNR=4.90dB; (b) group size=5, $\alpha = 0.5$, $b = 25$, SNR=10.12dB; (c) group size=100, $\alpha = 0.5$, $b = 2$, SNR=26.44dB. Compared with Fig.9, the reconstructed images for group size 5 and 100 are free from the influence of artefacts while keeping the similar SNRs.

The mixed sampling method can also eliminate the *flat area* in the convergence trend. For example, we here present the simulation results of different detector partition methods, as shown in Fig.17. Compared with Fig.10, the convergence trend is more smooth and it requires less iterations to achieve the precision limit.

E. 3D simulation

To demonstrate the performance in a 3D setting, we defined a 3D phantom of 128*128*128 voxels describing a 32*32*32 volume. The CCD detector plane was a square with side length 101 whose detector spacing was 0.5. To demonstrate how our method can be used for non-standard trajectories, we chose a randomised scanning path with 720 projection views, i.e. the point source location was randomly changed with a fixed radius r of 66 but with a random rotation. The rotation center was placed at the centre of the 3D object. The detector was always perpendicular to the line connecting the





(a) (b)

Fig. 18: Basic 3D simulation settings. (a) shows that the location of the point source is $x = r\sin a\cos b$,$y = r\sin a\sin b$,$z = r\cos a$, where a and b are randomly changed. The connection line of point source and the centre of the detector plane is always perpendicular to the detector plane. (b) is a slice of the 3D volume.

geometric centre of the detector and the point source, and the vertical distance between point source and detector plane was 132. The illustration of the scanning geometry and a slice of the 3D volume is shown in Fig.18. This simulation was performed on a computer with 48 Intel Xeon CPU cores and 256GB RAMs.

In this simulation, we did not divide the CCD area into sub-areas but simply treated each projection as a whole. We divided the volume into 2 sections for each dimension, thus we had 8 blocks in total. We set the group size to 90. b in β is set to 1, $\alpha = 0.5$ and $\gamma = 1$. In this situation the system matrix $\mathbf{A}$ was too big to be stored on the machine thus we only calculated the SNR term. The trend of SNR after different numbers of epochs is shown in Fig.19.

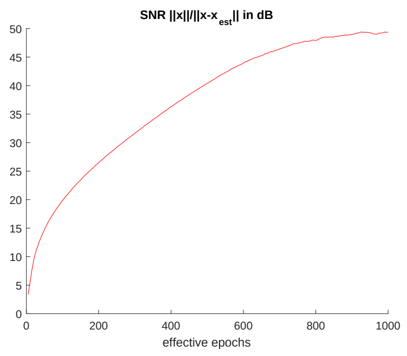


Fig. 19: The SNR trend of a 128*128*128 image when the group size is 90, $b = 1$, $\gamma = 1$ and $\alpha = 0.5$.

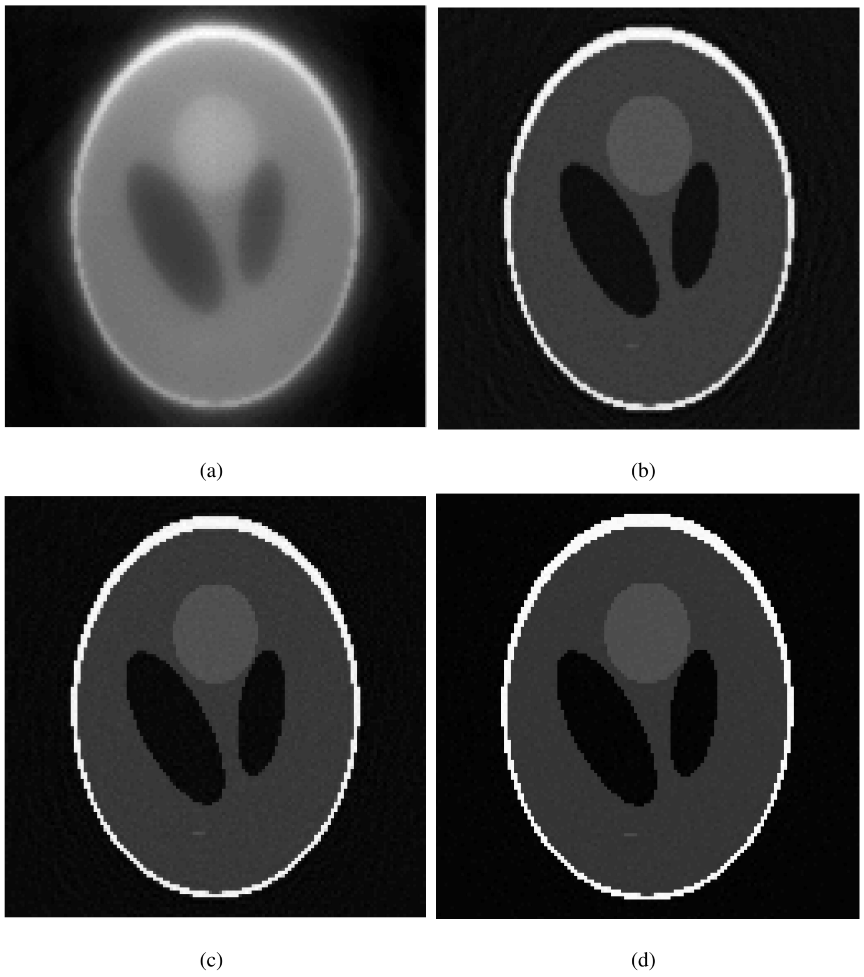


Fig. 20: The middle slice of the reconstructed image after 5(a),50(b),100(c),200(d) effective epochs.

We extracted one slice of the reconstructed image after different number of epochs to compare it to the theory one, as shown in Fig.20.

IV. Conclusion

Here we propose a parallel algorithm CSGD that is suitable for linear inverse problems $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ and enables additional flexibility in the way we can partition data sets in algebraic CT image reconstruction. As far as we know, CSGD is the first algorithm that combines row action methods and column action methods. Unlike other algorithms that demand the slice strategy of the reconstructed volume to be perpendicular to the rotation axial of the object to reduce data sharing between different computing nodes and a circular or helical scanning strategy, CSGD does not pose any restrictions on how to partition the reconstruction volume. It is thus applicable to generic scan trajectories. Furthermore, the CSGD method can be implemented using a server parameter parallel computing architecture. It thus has the potential to solve very large image reconstruction problems by using several computing nodes. The GCSGD method better utilises row information of the system matrix $\mathbf{A}$ and thus has faster convergence rates than CSGD. We have furthermore developed an importance sampling strategy. By non-uniformly selecting row blocks of $\mathbf{A}$ based on the projection area of the associated sub-volume, the sub-matrices of $\mathbf{A}$ with relatively higher density get chosen more often than those with higher sparsity, which increases the initial convergence rates. A mixed sampling strategy can be used to overcome the artefacts brought by importance sampling while keeping the fast convergence rate. We here present 2D simulation results which demonstrate the convergence properties of the proposed algorithms. 3D reconstruction results further verify the effectiveness of GCSGD in term of presenting visually acceptable reconstructions.

V. Open areas

Simulation results have shown that when the group size is fixed, there is a range for parameter b (or β) that guarantee the convergence of GCSGD. However, currently the choice on b is based on experience and trials, thus we want to develop a more systematic method, or an adaptive self-correction scheme for the determination of parameter b or β. Besides, in more realistic tomographic imaging settings, with noisy observations and an inaccurate system matrix, simply using current method, it is often difficult to obtain high quality reconstructions. As a result, incorporating regularisation terms into GCSGD is another important direction for our future work. We are also interested in applying the proposed algorithm in reality dataset under a distributed network and explore the properties and differences of synchronous and asynchronous communication strategies. In our paper we have tested the convergence property when $\gamma < 1$, which can be viewed as a form of asynchronous communication in a parallel computing architecture. More systematic investigations of this will be included in future work.

Reference

[1] Marcel Beister, Daniel Kolditz, and Willi A Kalender. Iterative reconstruction methods in x-ray ct. *Physica medica*, 28(2):94–108, 2012.

[2] Thomas M Benson, Bruno KB De Man, Lin Fu, and Jean-Baptiste Thibault. Block-based iterative coordinate descent. In *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*, pages 2856–2859. IEEE, 2010.

[3] Thomas M Benson and Jens Gregor. Framework for iterative cone-beam micro-ct reconstruction. *IEEE transactions on nuclear science*, 52(5):1335–1340, 2005.

[4] JR Bilbao-Castro, JM Carazo, JJ Fernández, and I Garcia. Performance of parallel 3d iterative reconstruction algorithms. *WSEAS Transactions on Biology and Biomedicine*, 1(1):112–119, 2004.

[5] Charles A Bouman and Ken Sauer. A unified approach to statistical tomography using coordinate descent optimization. *IEEE Transactions on image processing*, 5(3):480–492, 1996.

[6] Yair Censor. Row-action methods for huge and sparse systems and their applications. *SIAM review*, 23(4):444–466, 1981.

[7] Yair Censor. Parallel application of block-iterative methods in medical imaging and radiation therapy. *Mathematical Programming*, 42(1):307–325, 1988.

[8] Yair Censor, Paul PB Eggermont, and Dan Gordon. Strong underrelaxation in kaczmarz's method for inconsistent systems. *Numerische Mathematik*, 41(1):83–92, 1983.

[9] Yair Censor and Tommy Elfving. Block-iterative algorithms with diagonally scaled oblique projections for the linear feasibility problem. *SIAM Journal on Matrix Analysis and Applications*, 24(1):40–58, 2002.

[10] Yair Censor, Dan Gordon, and Rachel Gordon. Bicav: A block-iterative parallel algorithm for sparse systems with pixel-related weighting. *IEEE Transactions on Medical Imaging*, 20(10):1050–1060, 2001.

[11] Yair Censor, Dan Gordon, and Rachel Gordon. Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel computing*, 27(6):777–808, 2001.

[12] Junjun Deng, Hengyong Yu, Jun Ni, Lihe Wang, and Ge Wang. Parallelism of iterative ct reconstruction based on local reconstruction algorithm. *The Journal of supercomputing*, 48(1):1–14, 2009.

[13] Tommy Elfving. Block-iterative methods for consistent and inconsistent linear equations. *Numerische Mathematik*, 35(1):1–12, 1980.

[14] Tommy Elfving, Per Christian Hansen, and Touraj Nikazad. Convergence analysis for column-action methods in image reconstruction. *Numerical Algorithms*, pages 1–20, 2016.

[15] Alban Gervaise, Benoît Osemont, Sophie Lecocq, Alain Noel, Emilien Micard, Jacques Felblinger, and Alain Blum. Ct image quality improvement using adaptive iterative dose reduction with wide-volume acquisition on 320-detector ct. *European radiology*, 22(2):295–301, 2012.

[16] Jens Gregor and Thomas Benson. Computational analysis and improvement of sirt. *IEEE Transactions on Medical Imaging*, 27(7):918–924, 2008.

[17] Xue-Ping Guo. Convergence studies on block iterative algorithms for image reconstruction. *Applied Mathematics and Computation*, 273:525–534, 2016.

[18] Jiang Hsieh, Brian Nett, Zhou Yu, Ken Sauer, Jean-Baptiste Thibault, and Charles A Bouman. Recent advances in ct image reconstruction. *Current Radiology Reports*, 1(1):39–51, 2013.

[19] Alfredo N Iusem and Alvaro Rodolfo De Pierro. Convergence results for an accelerated nonlinear cimmino algorithm. *Numerische Mathematik*, 49(4):367–378, 1986.

[20] Filip Jacobs, Erik Sundermann, Bjorn De Sutter, Mark Christiaens, and Ignace Lemahieu. A fast algorithm to calculate the exact radiological path through a pixel or voxel space. *CIT. Journal of computing and information technology*, 6(1):89–94, 1998.

[21] Donghwan Kim and Jeffrey A Fessler. Parallelizable algorithms for x-ray ct image reconstruction with spatially non-uniform updates. *Proc. 2nd Intl. Mtg. on image formation in X-ray CT*, pages 33–6, 2012.

[22] JS Kole and FJ Beekman. Parallel statistical image reconstruction for cone-beam x-ray ct on a shared memory computation platform. *Physics in Medicine and Biology*, 50(6):1265, 2005.

[23] Taoran Li, Tzu-Jen Kao, David Isaacson, Jonathan C Newell, and Gary J Saulnier. Adaptive kaczmarz method for image reconstruction in electrical impedance tomography. *Physiological measurement*, 34(6):595, 2013.

[24] Deanna Needell and Joel A Tropp. Paved with good intentions: Analysis of a randomized block kaczmarz method. *Linear Algebra and its Applications*, 441:199–221, 2014.

[25] Jun Ni, Xiang Li, Tao He, and Ge Wang. Review of parallel computing techniques for computed tomography image reconstruction. *Current Medical Imaging Reviews*, 2(4):405–414, 2006.

[26] Willem Jan Palenstijn, Jeroen Bédorf, Joost Batenburg, M King, S Glick, and K Mueller. A distributed sirt implementation for the astra toolbox. 2015.

[27] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling i: Algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[28] Thomas Rodet, Frédéric Noo, and Michel Defrise. The cone-beam algorithm of feldkamp, davis, and kress preserves oblique line integrals. *Medical physics*, 31(7):1972–1975, 2004.

[29] Yoshiko Sagara, Amy K Hara, William Pavlicek, Alvin C Silva, Robert G Paden, and Qing Wu. Abdominal ct: comparison of low-dose ct with adaptive statistical iterative reconstruction and routine-dose ct with filtered back projection in 53 patients. *American Journal of Roentgenology*, 195(3):713–719, 2010.

[30] Zebang Shen, Hui Qian, Chao Zhang, and Tengfei Zhou. Accelerated variance reduced block coordinate descent. *arXiv preprint arXiv:1611.04149*, 2016.

[31] M Soleimani and T Pengpen. Introduction: a brief overview of iterative algorithms in x-ray computed tomography, 2015.

[32] Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262, 2009.

[33] Nate D Tang, Niels De Ruiter, JL Mohr, Anthony PH Butler, Philip H Butler, and R Aamir. Using algebraic reconstruction in computed tomography. In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, pages 216–221. ACM, 2012.

[34] Ge Wang, Hengyong Yu, and Bruno De Man. An outlook on x-ray ct research and development. *Medical physics*, 35(3):1051–1064, 2008.

[35] David W Watt. Column-relaxed algebraic reconstruction algorithm for tomography with noisy data. *Applied optics*, 33(20):4420–4427, 1994.

[36] Zhou Yu, Jean-Baptiste Thibault, Charles A Bouman, Ken D Sauer, and Jiang Hsieh. Non-homogeneous updates for the iterative coordinate descent algorithm. In *Electronic Imaging 2007*, pages 64981B–64981B. International Society for Optics and Photonics, 2007.

[37] Zhou Yu, Jean-Baptiste Thibault, Charles A Bouman, Ken D Sauer, and Jiang Hsieh. Fast model-based x-ray ct reconstruction using spatially nonhomogeneous icd optimization. *IEEE Transactions on image processing*, 20(1):161–175, 2011.