A Joint Row and Column Action Method for Cone-Beam Computed Tomography

Yushan Gao, Thomas Blumensath

***Abstract*—The inversion of linear systems is fundamental in Computed Tomography (CT) reconstruction. Computational challenges arise when trying to invert large linear systems, as limited computing resources mean that only part of the system can be kept in computer memory at any one time. In linear tomographic inversion problems such as x-ray tomography, even a standard scan can produce millions of individual measurements and the reconstruction of x-ray attenuation profiles typically requires the estimation of a million attenuation coefficients. To deal with the large data sets encountered in real applications and to efficiently utilise modern graphics processing unit (GPU) based computing architectures, combinations of iterative reconstruction algorithms and parallel computing schemes are increasingly applied. Whilst different parallel methods have been proposed, individual computations currently need to access either the entire set of observations or estimated x-ray absorptions, which can be prohibitive in many realistic applications. We present a fully parallelizable CT image reconstruction algorithm where each computation node works on arbitrary partial subsets of the data and the reconstructed volume. We further develop a non-homogeneously randomised selection criterion which guarantees that sub-matrices of the system matrix are selected more frequently if they are dense, thus maximising information flow through the algorithm. We compare our algorithm with block alternating direction method of multipliers (block ADMM) and show that our method is significantly faster for CT reconstruction.**

***Index Terms*—CT image reconstruction, parallel computing, gradient descent, coordinate descent, linear inverse problems.**

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) [1], [2] and the Feldkamp Davis Kress (FDK) [3], [4] method. However, in low signal to noise settings, if scan angles are under-sampled or if nonstandard trajectories are used, then less efficient, algebraic reconstruction methods can provide significantly better reconstructions [5]–[8]. These methods model the x-ray system as a linear system of equations [9]–[11]:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1c} \\ \vdots & \vdots & \vdots \\ a_{r1} & \dots & a_{rc} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_c \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ e_r \end{bmatrix}, \tag{1}$$

Manuscript received September 01, 2017. This work was supported by EPSRC grants EP/K029150/1 and EP/R002495/1, a University of Southampton PGR scholarship, a Faculty of Engineering and the Environment Lancaster Studentship and the China Scholarship Council.

Y. Gao and T. Blumensath are with the Faculty of Engineering and Environment, University of Southampton, Southampton, SO17 1BJ, UK (email: yg3n15@soton.ac.uk; Thomas.Blumensath@soton.ac.uk).

where $\mathbf{y} = [y_1, \cdots, y_r]^T, \mathbf{x} = [x_1, \cdots, x_c]^T$ and $\mathbf{e} = [e_1, \cdots, e_r]^T$ are projection data, reconstructed image vector and measurement noise respectively. The system matrix $A \in \mathbb{R}^{r \times c}$, with non-negative elements a_{cd} ($1 \leq c \leq r$,$1 \leq d \leq c$), can be computed by Siddon's method [12]. In reality, industrial CT is often used in cases where $\mathbf{y}$ and $\mathbf{x}$ can have millions of entries each [13] and where A, even though it is a sparse matrix, can have billions of entries. In these situations, solving Eq.1 directly by calculating A^{-1} is infeasible. More feasible approaches use A and A^T directly (instead of their inverse) to iteratively find an approximate solution of Eq.1.

There are many mature and efficient algorithms to solve Eq.1, including the conjugate gradient (CG) method and LSQR. These methods can be applied in small scale CT reconstructions when the system matrix A can be stored in computer memory [14], [15]. When A is too large to be kept in memory, it is often more efficient to re-compute it on the fly during each iteration, which can be done relatively efficiently using modern graphical processor unit (GPU). However, GPUs have limited internal memory. A thus has to be broken into smaller subsets so that the GPU only operates on a subset of the data at a time. Whilst it is possible to sequentially process all data in this fashion, this requires constant data transfer to the GPUs internal memory. An alternative is the use of optimisation algorithms that only work on subsets of the matrix A at any one time. The CG and LSQR methods, whilst having their own block forms [16], [17], do not work with a single subset of A per iteration. As a result, other algorithms for CT reconstructions have been proposed. 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 [18]–[22].

Row action methods divides the matrix A into several row blocks and the system equation thus becomes

$$\begin{bmatrix} \mathbf{y}_{I_1} \\ \vdots \\ \mathbf{y}_{I_M} \end{bmatrix} \approx \begin{bmatrix} A_{I_1} \\ \vdots \\ A_{I_M} \end{bmatrix} \mathbf{x}, \tag{2}$$

where $A_{I_i} \in \mathbb{R}^{m_i \times c}$ is the row block of system matrix A and $\mathbf{y}_{I_i} \in \mathbb{R}^{m_i \times 1}$ is the block of projection $\mathbf{y}$. The total block number is M and $\sum_{i=1}^{M} m_i = r$. The general iteration scheme in row action method can be summarised as

$$\mathbf{x}^k = \mathbf{x}^{k-1} + R(\mathbf{y}_I - A_I \mathbf{x}), \tag{3}$$

where $\mathbf{x}^k$ is the k^{th} iteration result, R is a relaxation matrix to control the iteration step length and $I \in \{I_i\}_{i=1}^{M}$. In CT reconstruction, most mature methods are row action methods. These

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

include the classical Kaczmarz family of algorithms (also known as the Algebraic Reconstruction Technique (ART)) [7], [23]–[25], the Simultaneous Algebraic Reconstruction Technique (SART) [19], [26], [27], the Simultaneous Iterative Reconstruction Technique (SIRT), [28]–[31] and component averaging (CAV) and its block form (BICAV) [32]–[34].

Column action methods, as a counterpart of row action method, divide the system matrix A into several column blocks. Eq.1 thus is divided as

$$\mathbf{y} \approx \begin{bmatrix} A^{J_1} & \cdots & A^{J_N} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{J_1} \\ \vdots \\ \mathbf{x}_{J_N} \end{bmatrix}, \tag{4}$$

where $A^{J_j} \in \mathbb{R}^{r \times n_j}$ is the column block of system matrix A and $\mathbf{x}_{J_j} \in \mathbb{R}^{n_j \times 1}$ is the block of reconstructed vector $\mathbf{x}$. The total block number is N and $\sum_{j=1}^{N} n_j = c$. The general iteration scheme in column action methods is of the form

$$\begin{aligned} \mathbf{x}_J^k &= \mathbf{x}_J^{k-1} + R\mathbf{r} \\ \mathbf{r} &= \mathbf{r} - A(\mathbf{x}_J^k - \mathbf{x}_J^{k-1}), \end{aligned} \tag{5}$$

where $J \in \{J_j\}_{j=1}^{N}$ and the initial $\mathbf{r} = \mathbf{y}$. The application of column action methods in CT reconstruction can be traced back to 1990s [35]–[37]. The preliminary column action method updates one voxel each time and it is easy to extend the preliminary method into group form [38]–[43]. Similar to row action methods, the selection criteria on which column blocks to update can be randomised either uniformly [44], [45] or based on specific probability [46], [47].

Row and column action methods also allow for parallel computation. For parallel row action methods, each processor (or node) needs to store the whole reconstructed image vector $\mathbf{x}$ since each update is on the entire image, whilst for column action methods, each node needs to store all of $\mathbf{y}$. As a result, the largest volume they can reconstruct is limited by the storage capacity of the computation nodes. An important exception to this is discussed in [48], where a row action method SIRT is discussed in which each node only requires parts of the reconstructed image vector $\mathbf{x}$. However, this method only works for circular scan trajectories and its scalability is limited due to the requirement that overlap between projections of adjacent image blocks should be small.

General sub-block methods are proposed in machine learning (ML). There has been interest in the development of methods that use more general updates, updating subsets of $\mathbf{x}$ with only subsets of $\mathbf{y}$ at a time [49], [50]. In this paper, these algorithms will be called "stochastic block coordinate descent" (SBCD). In particular, [51] and [52] independently proposed similar SBCD algorithms and both explored the application of variance reduction technique [53] to further accelerate the convergence rate. [54] proposed a semi-stochastic coordinate descent method to combine the stochastic gradient method and coordinate descent method to minimise a strongly convex problem. [55] mathematically proved the convergence rate of SBCD when the step length is decreasing and when the update strategy adopts a Gauss-Seidel type approach (updating the current column block depends on the previously updated column block), showing that the SBCD method has the same convergence rate as stochastic gradient methods when the objective function is convex. [50], [56] separately proposed the optimal sampling method in the SBCD method by randomly selecting column blocks and selecting row blocks based on a calculated probability.

The partition of A in SBCD is the same as our method. To define this, we partition A into $M \times N$ blocks. Let $I_i (1 \leq i \leq M)$ be an index set that indexes m_i ($\sum_{i=1}^{M} m_i = r$) rows in A and $J_j (1 \leq j \leq N)$ be an index set that indexes n_j ($\sum_{j=1}^{N} n_j = c$) columns in A . The matrix $A_{I_i}^{J_j}$ thus is a sub-matrix of A with row indexes I_i and column indexes J_j. Thus we can divide the linear system into $M \times N$ blocks:

$$\begin{bmatrix} \mathbf{y}_{I_1} \\ \vdots \\ \mathbf{y}_{I_M} \end{bmatrix} \approx \begin{bmatrix} A_{I_1}^{J_1} & \cdots & A_{I_1}^{J_N} \\ \vdots & \vdots & \vdots \\ A_{I_M}^{J_1} & \cdots & A_{I_M}^{J_N} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{J_1} \\ \vdots \\ \mathbf{x}_{J_N} \end{bmatrix} \equiv \begin{bmatrix} A_{I_1} \\ \vdots \\ A_{I_M} \end{bmatrix} \mathbf{x}. \tag{6}$$

Note that the index sets can be arbitrary partitions of the columns and rows and do not necessarily have to be consecutive. To facilitate the latter discussion, we also define residual $\mathbf{r} = \mathbf{y} - A\mathbf{x}$ and let block residual $\mathbf{r}_I$ be the subset of $\mathbf{r}$ defined as $\mathbf{y}_I - A_I\mathbf{x}$, where $I \in \{I_i\}_{i=1}^{M}$.

With this notation, the general update scheme is

$$\mathbf{x}_J = \mathbf{x}_J - \mu \cdot \nabla_J f_I(\mathbf{x}), \tag{7}$$

where $J \in \{J_j\}_{j=1}^{N}$, μ is the step length and $f_I(\mathbf{x}) = \|\mathbf{y_I} - A_I\mathbf{x}\|_2^2$, where the $\|\cdot\|$ is the l_2 norm of a vector. Thus the gradient $\nabla_J f_I(\mathbf{x}) = -2(A_I^J)^T(\mathbf{y}_I - A_I\mathbf{x})$.

There are two difficulties when applying SBCD in large scale CT reconstructions. The first one is that the step length μ is determined by calculating the Lipschitz constant of the gradient $\nabla_J f_I(\mathbf{x})$, which is computationally challenging. The second issue is that the computation of the gradient in SBCD requires the calculation of the block residue $\mathbf{r}_I = \mathbf{y}_I - A_I\mathbf{x}$, which needs the whole of $\mathbf{x}$. In other words, the system matrix A is not actually separable in column direction due to the need to calculate $\mathbf{r}_I$. This drawback makes the SBCD algorithm difficult to apply in a totally distributed network where each computation node only has partial access to both $\mathbf{x}$ and $\mathbf{y}$.

Another method that can be used for linear systems is the multisplitting (MS) method [57]. If MS method only partitions the A into column blocks [58], it always converges as long as A is of full column rank. However, If MS method partition the A into both row and column blocks. it becomes only applicable when A meets certain conditions. For example, when A is an "H-matrix" [59] or is a positive definite matrix [60]. The system matrix A in CT reconstruction does not meet these requirements and MS dividing A in both dimensions did not work in initial experiments we conducted for CT reconstruction.

Our CT reconstruction setting leads to a convex optimization problem. The alternating direction method of multipliers (ADMM) is a popular tool in the parallelization of convex optimization probelms as it allows the decomposition into several smaller sub-problems [61]. Several versions of ADMM were designed to work on small subsets of either $\mathbf{x}$ *or* $\mathbf{y}$ [62]. In [63], a block ADMM method was proposed that works by breaking the problems into small subsets of $\mathbf{x}$ *and*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

$\mathbf{y}$. This version of block ADMM introduces three sets of additional nuisance variables and each sub problem requires the solution to a smaller linear inverse problem, that can often be solved approximately using a small number of CG iterations. The new nuisance variables increase the overall storage requirements and the requirement for each node to solve a linear inverse problem in each iteration leads to relatively slow convergence and a high computation burden in large scale CT reconstruction, which will be illustrated in our paper.

II. Introduction of CSGD

In this paper, we consider large scale CT reconstruction problems in a distributed networks, where each computation node only has limited access to both $\mathbf{y}$ and $\mathbf{x}$. Inspired by SBCD and block ADMM, our goal here is to develop a parallelisable algorithm that works with generic x-ray tomographic scanning trajectories and that is faster and more memory efficient than block ADMM. Our novel algorithm, called coordinate-wise stochastic gradient descent (CSGD), also introduces nuisance variables, but we use fewer variables than block ADMM. Furthermore, CSGD simplifies the step length calculation as well as the residue update scheme of SBCD type algorithms and thus enables a full column decomposition for A. The algorithm 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.

A. Derivation of the algorithm

The proposed CSGD is similar to SBCD. The main goals of CSGD are to find a simpler strategy to compute the step length μ and to efficiently approximate the residue $\mathbf{r}_I$ without having to compute the product $A_I\mathbf{x}$ in each iteration.

Similar to SBCD, after selecting a row block $I \in \{I_i\}_{i=1}^M$, CSGD operates on the object function

$$f_I(\mathbf{x}) = \|\mathbf{y}_I - A_I\mathbf{x}\|^2, \tag{8}$$

with gradient

$$\mathbf{g} = \nabla f_I(\mathbf{x}) = -2(A_I)^T(\mathbf{y}_I - A_I\mathbf{x}) = -2(A_I)^T\mathbf{r}_I. \tag{9}$$

A coordinate descend update scheme is adopted and only those elements whose indices are in the set $J \in \{J_j\}_{j=1}^N$ are updated, so that the descent operator is

$$\widetilde{\mathbf{g}} = \begin{bmatrix} \mathbf{g}_J \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} (A_I^J)^T\mathbf{r}_I \\ \mathbf{0} \end{bmatrix}. \tag{10}$$

Along with this new modified direction, the update on voxels becomes

$$\mathbf{x}^k = \mathbf{x}^{k-1} + \mu\widetilde{\mathbf{g}} \rightarrow \begin{cases} \mathbf{x}_J^k & = \mathbf{x}_J^{k-1} + \mu\mathbf{g}_J \\ \mathbf{x}_{\widehat{J}}^k & = \mathbf{x}_{\widehat{J}}^{k-1}, \end{cases} \tag{11}$$

where μ is the gradient step length, $\widehat{J}$ is the complement to the set J and $\mathbf{g}_J$ is a sub set of $-\frac{1}{2}\mathbf{g}$. The steepest descent idea is to make the direction of $\nabla f_I(\mathbf{x}^k)$ perpendicular to the direction of $\widetilde{\mathbf{g}}$, i.e.

$$((A_I)^T(\mathbf{y}_I - A_I\mathbf{x}^k))^T\widetilde{\mathbf{g}} = 0. \tag{12}$$

Use the fact that $x^k = x^{k-1} + \mu\widetilde{\mathbf{g}}$ and $A_I\widetilde{\mathbf{g}} = A_I^J\mathbf{g}_J$, the μ leading to the maximum descent is

$$\mu = \frac{{\mathbf{g}_J}^T(A_I^J)^T\mathbf{r}_I}{{\mathbf{g}_J}^T(A_I^J)^TA_I^J\mathbf{g}_J} = \frac{{\mathbf{g}_J}^T\mathbf{g}_J}{{\mathbf{g}_J}^T(A_I^J)^TA_I^J\mathbf{g}_J}. \tag{13}$$

This calculation does not require the corresponding computation node to have access to the whole row or column block of matrix A and does not have to calculate the Lipshitz constant to determine the step length. When the matrix A cannot be stored and need to be generated on the fly, CSGD iteration only requires to use a sub matrix A_I^J and its transpose. Furthermore, GPUs can calculate the forward projection $A_I^J\mathbf{g}_J$ and backward projection $(A_I^J)^T\mathbf{r}_I$ very efficiently to achieve GPU-accelerated projection operations [64], [65].

The step size derived from Eq.13 is chosen to reduce $\|\mathbf{r}_I\|$ instead of $\|\mathbf{r}\|$. We observed experimentally that our choice of μ can lead to instability in the algorithm. One method to avoid this is to introduce an additional relaxation parameter $\beta < 1$, which led CSGD to converge to a precision level similar to that of SIRT and CAV. The alternative of using a constant step length μ also led to convergence, but required a careful choice of step length (thus making parameter tuning difficult) and led to relatively slow convergence.

$\mathbf{r}_I$ plays an important role in CSGD since it determines the update direction of $\mathbf{x}$. If we had access to all $\mathbf{x}_j$, then we could simply compute $\mathbf{r}_I = \mathbf{y}_I - \sum_j \mathbf{x}_j$. In our setting, the node only has access to one $\mathbf{x}_j$, thus the above computation is not possible. Instead, each node computes a quantity $\mathbf{z}_I^j = \mathbf{A}_I^{J_j}\mathbf{x}_{J_j}$. However, as discussed later, we might not compute all $\mathbf{z}_I^j$ in each iteration. In this case, we use an older versions of $\mathbf{z}_I^j$ to approximate $\mathbf{r}_I = \mathbf{y}_I - \sum_{j=1}^N \mathbf{z}_I^j$. For smoothly varying cost functions, using old residual information when calculating gradients is similar to the use of old gradients in parallel methods that can often be shown to converge [66], [67].

The pseudo-code of the basic computation blocks is shown in the Fig.1.

Algorithm 1 The algorithm for the computation performed at each node

Initialization: select system matrix's row index $I \in \{I_i\}_{i=1}^M$ and column index $J \in \{J_j\}_{j=1}^N$
$\mathbf{g}_J = (A_I^J)^T\mathbf{r}_I$
$\mu = \beta\frac{(\mathbf{g}_J)^T\mathbf{g}_J}{(\mathbf{g}_J)^T(A_I^J)^TA_I^J\mathbf{g}_J}$
$\mathbf{x}_J^k = \mathbf{x}_J^{k-1} + \mu\mathbf{g}_J$
$\mathbf{z}_I^j = A_I^J\mathbf{x}_J^k$

To parallelize the algorithm, we do not have to update all sub-blocks before updating $\mathbf{r}$. We define percentages α and γ to specify the faction of row and column blocks to be updated. The algorithm is shown in Algo.2. In the following discussion, when operations from line 4 to line 16 are performed for one time, the algorithm is said to perform for "one epoch".

B. Comparison CSGD with block ADMM

Block ADMM has the same parallel computing architecture as CSGD. To facilitate the comparison, we briefly present the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

Algorithm 2 CSGD algorithm which parallelize both row and column blocks.

Input: $\mathbf{y}$, α, β, γ, $\{I_i\}_{i=1}^M$ and $\{J_j\}_{j=1}^N$.
Initialisation: $\mathbf{x}=\mathbf{0}$, (i.e., $\{\mathbf{x}_{J_j}\}_{j=1}^N=\mathbf{0}$), $\{\hat{\mathbf{x}}^i\}_{i=1}^M=\mathbf{x}$, $\{\mathbf{z}^j\}_{j=1}^N=\mathbf{0}$ and $\mathbf{r}=\mathbf{y}$.
while stopping criterion is not met **do**
  for $\tilde{j}$=1,2,...,γN **in parallel** (J loop) **do**
    randomly draw J from $\{J_j\}_{j=1}^N$ with replacement
    $\hat{\mathbf{x}}^i(1\le i\le M)=\mathbf{0}$
    for $\tilde{i}$=1,2,...,αM **in parallel** (I loop) **do**
      randomly draw I from $\{I_i\}_{i=1}^M$ with replacement
      $\mathbf{g}_J=(A_I^J)^T\mathbf{r}_I$
      $\mu_j^i=\beta\frac{(\mathbf{g}_J)^T\mathbf{g}_J}{(\mathbf{g}_J)^T(A_I^J)^TA_I^J\mathbf{g}_J}$
      $\hat{\mathbf{x}}_J^i=\mathbf{x}_J+\mu_j^i\mathbf{g}_J$
      $\mathbf{z}_I^j=A_I^J\hat{\mathbf{x}}_J^i$
    end for
  end for
  $\mathbf{r}=\mathbf{y}-\sum_j\mathbf{z}^j$
  for all updated J, $\mathbf{x}_J=\sum_i(\hat{\mathbf{x}}_J^i)/(\alpha M)$
end while
$\mathbf{x}_{solution}=[\mathbf{x}_{J_1},...,\mathbf{x}_{J_N}]^T$

basic operation of block ADMM algorithm [63], as shown in Algo.3. Here **avg** is the element-wise averaging operator. The

Algorithm 3 Block ADMM iteration.

Input: $\mathbf{y}$, ρ, $\{I_i\}_{i=1}^M$ and $\{J_j\}_{j=1}^N$.
Initialisation: $\mathbf{x}=\tilde{\mathbf{x}}=\mathbf{0}$, $\mathbf{z}=\tilde{\mathbf{z}}=\mathbf{0}$, $\{\mathbf{x}^i\}_{i=1}^M=\mathbf{0}$, $\{\mathbf{z}^j\}_{j=1}^N=\mathbf{0}$
for k=1,2,...,k_{max} **do**
  $\mathbf{x}^{k+\frac{1}{2}}=\mathbf{x}^k-\tilde{\mathbf{x}}^k$
  $\mathbf{z}^{k+\frac{1}{2}}=\frac{1}{1+\frac{\rho}{2}}\mathbf{y}+\frac{1}{\frac{2}{\rho}+1}(\mathbf{z}^k-\tilde{\mathbf{z}}^k)$
  for $\tilde{j}$=1,2,...,N **in parallel** (J loop) **do**
    randomly draw J from $\{J_j\}_{j=1}^N$ with replacement
    for $\tilde{i}$=1,2,...,M **in parallel** (I loop) **do**
      randomly draw I from $\{I_i\}_{i=1}^M$ with replacement
      $({\mathbf{x}_J^i}^{k+\frac{1}{2}},{\mathbf{z}_I^j}^{k+\frac{1}{2}})=\Pi_{A_I^J}(\mathbf{x}_J^k-(\tilde{\mathbf{x}}_J^i)^k,{\mathbf{z}_I^j}^k+\tilde{\mathbf{z}}_I^k)$
    end for
  end for
  $\mathbf{x}^{k+1}=\mathbf{avg}(\mathbf{x}^{k+\frac{1}{2}},\{{\mathbf{x}^i}^{k+\frac{1}{2}}\}_{i=1}^M)$
  $(\mathbf{z}^{k+1},\{{\mathbf{z}^j}^{k+1}\}_{j=1}^N)=\mathbf{exch}(\mathbf{z}^{k+\frac{1}{2}},\{{\mathbf{z}^j}^{k+\frac{1}{2}}\}_{j=1}^N)$
  $\tilde{\mathbf{x}}^{k+1}=\tilde{\mathbf{x}}^k+\mathbf{x}^{k+\frac{1}{2}}-\mathbf{x}^{k+1}$
  $\tilde{\mathbf{x}}^i)^{k+1}=\tilde{\mathbf{x}}^i)^k+{\mathbf{x}^i}^{k+\frac{1}{2}}-\mathbf{x}^{k+1}$
  $\tilde{\mathbf{z}}^{k+1}=\tilde{\mathbf{z}}^k+\mathbf{z}^{k+\frac{1}{2}}-\mathbf{z}^{k+1}$
end for
$\mathbf{x}_{solution}=\mathbf{x}^{k_{max}+\frac{1}{2}}$

Π projection is a linear operator

$$\Pi_{A_I^J}(c,d)=\begin{bmatrix}I_d & (A_I^J)^T\\ A_I^J & -I_d\end{bmatrix}^{-1}\begin{bmatrix}I_d & (A_I^J)^T\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}c\\ d\end{bmatrix} \quad (14)$$

where I_d is an identity matrix whose size is determined by the size of A_I^J. To solve this equation, the CG method can be used. Standard techniques to speed up block ADMM include the early termination of the CG iteration and a variable ρ-update scheme [61]. The exchange operator $\mathbf{exch}(c,\{c_j\}_{j=1}^N)$ is given by

$$\begin{aligned}\mathbf{z}_{I_i}^j&=c_j+\frac{(c-\sum_{j=1}^N c_j)}{N+1}\\ \mathbf{z}_{I_i}&=c-\frac{(c-\sum_{j=1}^N c_j)}{N+1}\end{aligned} \quad (15)$$

When applying block ADMM and CSGD in a distributed network, they share the same architecture. For $M=N=2$, parallel block ADMM is shown diagrammatically in Fig.1. Here, each computation node (ovals) stores one image block

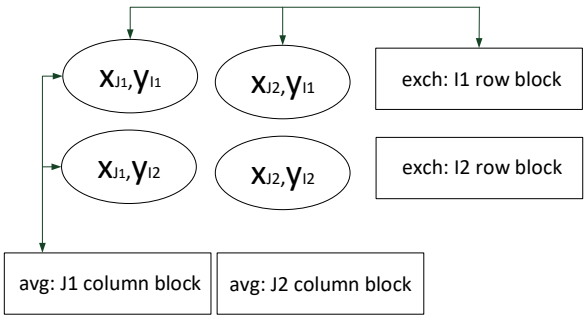


Fig. 1. One parallel implementation of block ADMM. Computation nodes (ovals) compute Π projection and master nodes (square boxes) compute "**avg**" and "**exch**".

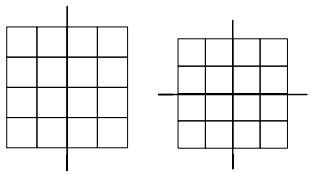
$\mathbf{x}_J$ ($J\in\{J_j\}_{j=1}^N$) and one data block $\mathbf{y}_I$ ($I\in\{I_i\}_{i=1}^M$). There are two types of master node (square boxes). One type performs the "**avg**" procedure for column blocks and the other type performs the "**exch**" procedure for row blocks. One set of master nodes stores $\mathbf{x}_J^{k+\frac{1}{2}}$, $\mathbf{x}_J^k$ and $\tilde{\mathbf{x}}_J^k$ and the other set of master nodes stores $\mathbf{y}_I^{k+\frac{1}{2}}$, $\mathbf{y}_I^k$ and $\tilde{\mathbf{y}}_I^k$. The same distributed architecture of Fig. 1 is also suitable for CSGD with $\alpha=\gamma=1$. For each J, CSGD requires the exchange and summation of $\mathbf{z}_I^j$ (i.e. over columns in Fig.1) and for each I, CSGD requires the exchange and summation of $\hat{\mathbf{x}}_J^i$ (i.e. over rows in Fig.1). These operations are similar to the "**avg**" and "**exch**" procedure of ADMM respectively. Note that these operations can be implemented efficiently using a message passing approach.

The storage demand is different between block ADMM and CSGD. Overall, CSGD requires storage of vectors $\mathbf{y}\in\mathbb{R}^{r\times1}$, $\mathbf{r}\in\mathbb{R}^{r\times1}$ and $\mathbf{x}\in\mathbb{R}^{c\times1}$ and of the set of N vectors $\{\mathbf{z}^j\}_{j=1}^N\in\mathbb{R}^{r\times1}$, so the total memory requirement for CSGD is $(N+2)r+c$. Block ADMM requires more storage, as it requires storage of $\mathbf{x}$, $\tilde{\mathbf{x}}$, $\mathbf{z}$, $\tilde{\mathbf{z}}$, $\mathbf{y}$ and the M vectors $\{\mathbf{x}^i\}_{i=1}^M$ and $\{\tilde{\mathbf{x}}^i\}_{i=1}^M$ and the N vectors $\{\mathbf{z}^j\}_{j=1}^N$, leading to a total storage demand of $(N+3)r+(2M+2)c$.

C. Partition methods in CT case

In our CT reconstruction problem, the partition of A along columns is equivalent to the partition of the image. Two examples of column partition are 1) to cut the image along one dimension and 2) to cut the image along all dimensions. Two 2D examples are shown in Fig.2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging



(a) (b)

Fig. 2. (a) shows the partition that along with one dimension for a 16-pixel image, (b) shows a partition that along both image dimensions.

To understand how partitioning row blocks of A relates to the tomographic imaging setting, we take the 2D scanning model shown in Fig.3 as an example. When detector and source are at a given location, we obtain projections along a range of paths from the source to the different detector elements. For one source/detector location, these measurements will be called one "projection". We could form the row blocks of A with random projections, but then keep them fixed during different epochs. A non-random version of this will be called "deterministic partitioning" in which one row block contains sequential projections from successive projection angles. We can further divide the detector into several sub-areas and treat projections from each sub-area as a "sub-projection". An illusion of this in 3D is shown in Fig.4, where one projection is divided into 4 sub-projections. When forming the row blocks, different sub-projections from different projection angles can be grouped together. This will be seen to be advantageous in the importance sampling strategy discussed next.

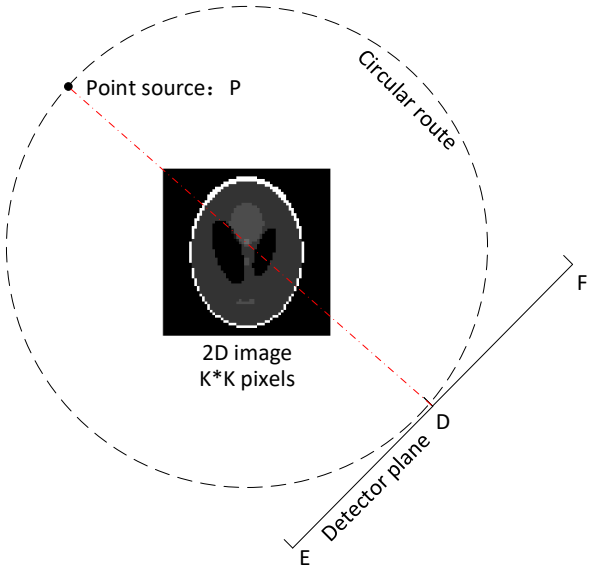
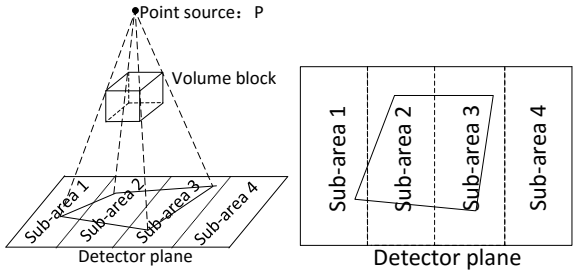


Fig. 3. A standard 2D scanning geometry with a Shepp-Logan phantom, where the P is the x-ray source, O is the centre of the object and the rotation centre. D is the centre of the detector. Source and detector rotate around the centre and take measurements at different angles. The linear detector is evenly divided into to sub-areas DE and DF, which will be used in importance sampling discussed later.

D. Importance sampling

When slicing the detector into several sub-areas, we can also form row blocks dynamically. It means that the sub-areas forming row blocks are changing for different epochs. Based on the proposed algorithm, it is straightforward to develop a random sampling strategy that goes through all projection views and all detector sub-areas arbitrarily. Looking at Algo.2, for one image block $\mathbf{x}_{J_j}(1 \leq j \leq N)$, we could randomly select sub-areas to form one row block $I_i(1 \leq i \leq M)$, with each sub-areas being chosen with equal probability as long as the sub-area receives x-rays passing through $\mathbf{x}_{J_j}$. This sample method will be called "uniform sampling". Considering the sparsity of A, we further develop an importance sampling strategy. Fig.4 illustrates that the projection of a sub-image block only intersects a small part of the detector. When sampling sub-projections for each sub-image block $\mathbf{x}_{J_j}(1 \leq j \leq N)$, a vector P_j representing the probability of choosing each detector sub-area is calculated. The calculation is based on projection areas of $\mathbf{x}_{J_j}$ on each sub-detector area at different projection angles. Then $\alpha * 100\%$ of the sub-areas are sampled with probability P_j and are grouped into αM row blocks. These row blocks are assigned to different nodes to perform line 9-12 in Algo.2. In this case, the detector sub-areas receiving more projections from the current sub-image block $\mathbf{x}_{J_j}$ are more likely to be chosen.



(a) (b)

Fig. 4. (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.

E. 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 network of computation nodes[1]. Computation nodes produce two outputs, as displayed in line 11 and line 12 in Algo.2. These are then either sent to larger, but slow storage or directly to other nodes, where they are eventually used to compute line 15 and line 16 in Algo.2, which can be performed efficiently using message passing interface reduction methods.

Three aspects affect performance:

1) Computational complexity in terms of multiply-adds.
2) Data transfer requirements between data storage and a processing unit as well as between different processing units.

[1]A serial version running on a single computation 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

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 properties:

1) Computational complexity is dominated by the computation of matrix vector products involving A_I^J and its transpose, especially as 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 computation nodes to receive $\mathbf{r}_I$ and $\mathbf{x}_J$ and transmit $\hat{\mathbf{x}}_J^i$ and $\mathbf{z}_I^j$. 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

The simulations are divided into two parts, the first part explores the performance of CSGD in CT reconstruction. The second part compares CSGD with block ADMM.

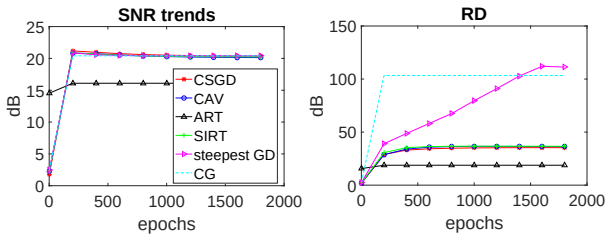
A. CSGD in CT reconstruction

We used two criteria to evaluate the performance. 1) signal to noise ratio (SNR): $20log_{10}(\|\mathbf{x}_{true}\|/\|\mathbf{x}_{true} - \mathbf{x}_{est}\|)$, 2) relative distance (RD): $20\log_{10}(\|\mathbf{x}_{lsq}\|/\|\mathbf{x}_{lsq} - \mathbf{x}_{est}\|)$. The $\mathbf{x}_{true}$, $\mathbf{x}_{est}$ and $\mathbf{x}_{lsq}$ are the true phantom image vector, the reconstructed image vector and the least square solution respectively.

In the following simulations, we used the Shepp-Logan phantom and, unless stated otherwise, K in Fig.3 is 16 and the side length of each pixel is 1mm, the length of OP and OD are always 100mm. The rotation interval for source and detector is 10°. The detector contains 30 pixels of size 1mm. The $\mathbf{e}$ in Eq.1 is Gaussian white noise with variance σ of 0.1 to the simulated observations. The SNR of projection data $\mathbf{y}$ ($20log_{10}(\|\mathbf{y}\|/\|\mathbf{e}\|)$) is 25.8 dB. The default partition method of projection data $\mathbf{y}$ uses the static type "deterministic partitioning". The default partition in the image domain uses the method shown in Fig.2a.

B. Comparison to other methods

We start by comparing our method to a range of other algorithms popular in CT reconstruction. For CSGD we used $M = 8$, $N = 4$, $\beta = 0.23$ and $\alpha = \gamma = 1$. We compared our method to CG, steepest gradient descent (GD), SIRT, ART and CAV. The results are shown in Fig.5. It can be seen that for noisy observations, the CSGD and the other method (except for ART method) all obtains nearly the same SNR. However, whilst CG and steepest GD converge to the least square solution, CSGD, SIRT, ART and CAV do not. This indicates that similar to SIRT, ART and CAV, CSGD also iterates the $\mathbf{x}$ towards to a weighted least square solution rather than the least square solution itself (see our fixed point analysis in the Appendix).

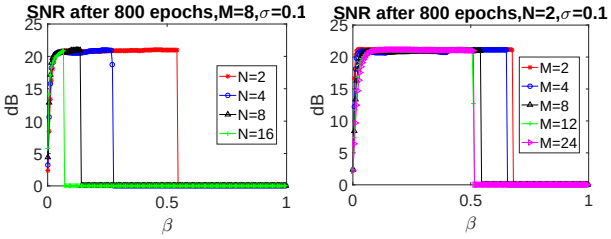


(a) (b)

Fig. 5. Comparison of CSGD with other methods. The performance of CSGD, including the final SNR and RD level, is similar to the SIRT and CAV methods.

C. Influence of algorithm parameters

Our algorithm has three parameters that have to be set, α, β and γ. Due to space constraints we only discuss α and β and set the $\gamma \equiv 1$. Smaller γ lead to similar results to those observed with smaller α. We start by an empirical evaluation of β. To determine the range of suitable parameters β, we explored the performance by varying M, N and β. To show the largest value of β that can be used for different values of N and M, we run the algorithm for 800 epochs for different values of β. The results are shown in Fig. 6 where we see that the largest β value leading to highly accurate solutions is approximately $\frac{1}{N}$. Increasing N reduces the β range, and there is also a relationship on M which is less clear. We next turn



(a) (b)

Fig. 6. SNR value after 800 epochs for different values of β and for different N and M. Increasing the M or N reduces the acceptable β range and the range is influenced more by N value instead of M. In this simulation, the $\alpha \equiv 1$.

to the influence of the α. Reducing α reduces the computation of CSGD within one epoch as only some of the updates are computed. We set M=12 and N=2 and generated the data as before. Simulation results show that reducing α increase the acceptable β range, as shown in Fig.7.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

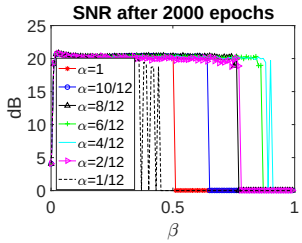
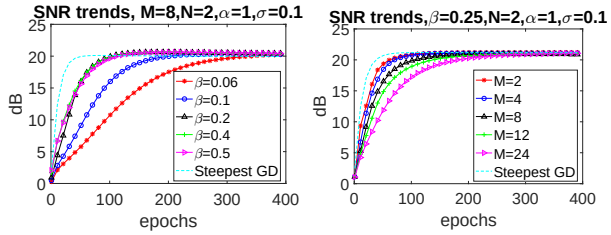


Fig. 7. When $M = 12, N = 2$, the β value range for different α. Unless that α is too small ($\frac{1}{12}$), decreasing α generally enlarges the range of acceptable β.

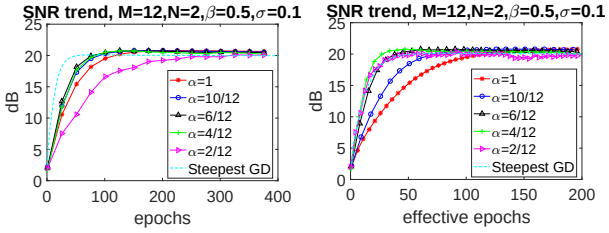
Convergence speed is influenced by α, β, M and N. In general, convergence is slower for larger values of N and M and for smaller β and α. The comparisons of β and M are shown in Fig. 8. It can be seen that in general, increasing β was found to lead to faster convergence, up to a point where the algorithms started to diverge. When the β is constant, increasing the M (or N, which is not provided here) slows down the convergence speed. We here also provide the steepest GD results as a reference to show that CSGD can obtain the same precision level as steepest GD does.



(a) (b)

Fig. 8. (a) shows that within the acceptable β range, increasing the β is found to lead to faster convergence, up to a point where the algorithms is about to diverge. (b) shows that when N is fixed, increasing M slows down the total convergence rate.

Before discussing the influence on convergence speed caused by α, we introduce a new concept "effective epoch". The number of sub-matrices $A_{I_i}^{J_j}$ (in Eq.6) that CSGD used per epoch is proportional to the parameter α. As the computational speed is dominated by matrix vector products, when we compare the difference in convergence speed for methods using different αs (for example, $\alpha = 1$ and $\alpha = 0.5$), we took account of the reduction in computational effort when using smaller α. The epoch count is multiplied by α and is called as the *effective epoch*. As a result, for different α, the computation amount after one "effective epoch" is the same with each other. We need to point out that the "effective epoch" does not consider the data transfer influence. When α is small, one "effective epoch" includes more epochs, which means more data transfer between nodes. Here we mainly focus on cases when the data transfer is much more efficient than calculating matrix-vector multiplications and thus is negligible. The comparison of convergence speed with different α is shown in Fig.9. Interestingly, by looking at effective epochs, we see that CSGD can converge faster than gradient descent for smaller α.



(a) (b)

Fig. 9. (a) shows that when β is fixed, reducing the α to around 0.5 increases the convergence speed. However, when the α is reduced too small as $\frac{2}{12}$, the convergence speed is slowed down. (b) shows that from "effective epoch " point of view, reducing the α is always helpful to increase the convergence speed and the increased convergence speed can be similar to the steepest GD when $\alpha \leq 0.5$.

D. Partitioning the image and the detector

In this section, larger simulations are conducted. The CT scanning geometry still uses the form shown in Fig.3, but with $K = 64$. The length of OP and OD is set to 115mm. The rotation interval for the point source and the detector is 3°. The detector contains 130 pixels. The $\mathbf{e}$ in Eq.1 is Gaussian white noise with variance σ of 1 to the simulated observations. The SNR of projection data $\mathbf{y}$ ($20log_{10}(\|\mathbf{y}\|/\|\mathbf{e}\|)$) is 33.8 dB. In the following section, the partition of the image follows Fig.2b. We here compare the partition of the linear detector into one or two areas, as shown in Fig.3. We also compare uniform sampling and importance sampling. The algorithm is performed for 20 epochs and the SNR value for different β values at this point is shown in Fig.10 (a)-(b) and two typical SNR trend in Fig.10 (c)-(d). Under different α, the uniform sampling strategy always obtains higher SNR after the same epochs than deterministic partitioning. Besides, the uniform sampling method has an even wider β range than deterministic partitioning, which is easier for parameter tuning. The uniform sampling method can be further improved when the detector is sliced into two sub-areas. This method avoids choosing the sub-detector areas which do not receive any projections from the currently selected sub-image block. However, when $\alpha = 1$, the two sub-detector situation does not increase the convergence rate. This is because that those sub-detectors which do not receive projections from the current sub-block still have to be selected since the α is too large. When the α decrease to a value (e.g. $\frac{6}{24}$) that guarantees all selected sub-detectors are those who receive projections, slicing the detector area into two sub-areas obtains higher SNR than not slicing situation. Furthermore, when α is small, the importance sampling strategy can obtain higher SNR than uniform sampling method at the initial iterations.

E. Comparison of CSGD and block ADMM

As discussed previously, block ADMM allows for the same partitioning of A. To compare CSGD and block ADMM, a random system matrix $A_{256\times128}$ and a random vector $\mathbf{x}_{128\times1}$ are used to reduce the computation requirements in our simulations. The linear system is a noise free model here.

As each CSGD iteration and each block ADMM iteration require different amounts of computational effort, we here do

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCI.2018.2857446, IEEE Transactions on Computational Imaging

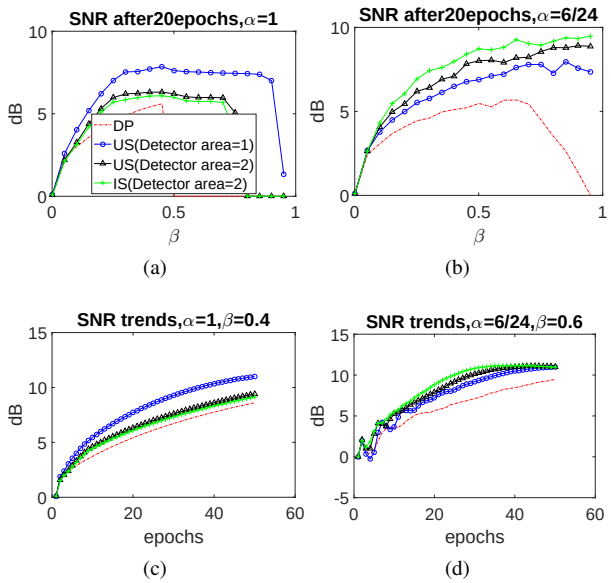


Fig. 10. Comparisons of different sample methods. The $M = 24, N = 2, \sigma = 1$."DP" is deterministic partitioning (defined in II-C), "US" is uniform sampling method (defined in II-D) and "IS" is importance sampling method (defined in II-D). The uniform sample or importance sampling both show faster convergence speed and wider acceptable β range than previous deterministic partitioning. Importance sampling methods outperform the others when α is $\frac{6}{24}$.

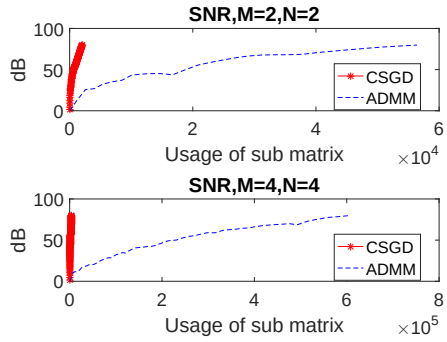


Fig. 11. Under noise free assumption, CSGD uses much fewer matrix-vector multiplications to achieve predefined SNR. The β used in CSGD is $\frac{1}{2N}$.

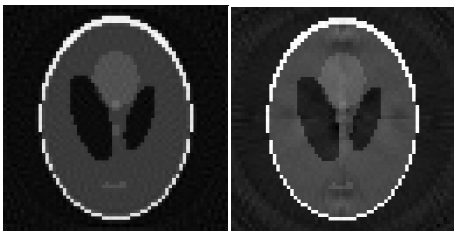


Fig. 12. Comparison of reconstruction results using CSGD (a) and block ADMM (b). Although the SNR of both images is the same, (b) shows much clearer inner artefacts than (a).

not compare SNR after each epoch. Instead, we plot SNR against the number of times the algorithm has computed a matrix-vector product involving the sub-matrices of A, as this is the dominating computational cost here. Both CSGD and block ADMM use all subsets of the matrix A and are stopped when their solutions reach 80 dB SNR. For block ADMM, we determined the best parameter values for variable ρ and also stopped the CG method after as few iterations as possible to allow algorithm convergence to the required level.

The convergence comparison for CSGD and block ADMM are shown in Fig.11. The results demonstrate the significant speed advantage offered by CSGD, which requires significantly fewer matrix-vector multiplications compared to block ADMM. Furthermore, the fully distributed form of CSGD used here, i.e. $\alpha = 1$, is not the optimal use of CSGD. According to the previous simulations, setting $\alpha < 1$ can further increase convergence speed.

We have also compared CSGD and block ADMM on the CT simulation of the previous simulation data with $K = 64$ and a detector with 130 pixels. The image was partitioned again as in Fig.2b. Both methods were stopped once the SNR of the reconstructed image had reached 20 dB. The reconstructed images under noise free situation are shown in Fig.12. Although the SNR of both images is the same, the ADMM reconstruction in Fig.12b shows much clearer artefacts along with the boundaries of adjacent sub-image blocks whilst the CSGD results in Fig.12a do not show these effects.

IV. Conclusion

CSGD was designed for a distributed reconstruction of cone beam CT data under arbitrary scan trajectories. In the distributed network, each node is assumed to have limited storage capacity and thus all nodes operate with limited access to the projection data and reconstructed volume. Whilst the method does not converge to the least squares solution, the solution is found empirically to be comparable in quality to those found with other common tomographic reconstruction algorithms, such as SIRT and CAV, but at a significant computational advantage. The parallel architecture is the same as that of block ADMM, which is a general algorithm for separable convex optimization. However, for large scale CT reconstruction, block ADMM is less attractive compared to CSGD. One advantage of CSGD is that it requires less storage compared to block ADMM. Another significant advantage of CSGD is that it converges with significantly fewer matrix vector products as it avoids the calculation of matrix inverses. This means that CSGD converges much faster compared to block ADMM.

We have furthermore developed an importance sampling strategy, that has been shown to further increase initial convergence. A theoretical analysis of the algorithm's convergence properties is ongoing and so is the inclusion of regularisation terms in the method.

Appendix

Whilst we do not have a formal convergence proof of CSGD yet, it is instructive to analyse the fixed points of the algorithm.

We here look at the deterministic version of the algorithm with $\alpha = 1$.

The algorithm updates two quantities, $\mathbf{x}$ and $\mathbf{z}$. Let $(\mathbf{x}^{k+1}, \mathbf{z}^{k+1}) = T(\mathbf{x}^k, \mathbf{z}^k)$ define one iteration of the algorithm. Let $\mathbf{x}^\star$ and $\mathbf{z}^\star$ be fixed points of the operator $T(\mathbf{x}, \mathbf{z})$ defined by $(\mathbf{x}^\star, \mathbf{z}^\star) = T(\mathbf{x}^\star, \mathbf{z}^\star)$. Similar results to the once derived here for the deterministic algorithm can also be obtained for the randomised versions and for $\alpha < 1$ if we look at points for which $(\mathbf{x}^\star, \mathbf{z}^\star) = \mathbb{E}\{T_r(\mathbf{x}^\star, \mathbf{z}^\star)\}$, where $\mathbb{E}\{\cdot\}$ is the expectation with respect to the random iteration operator $T_r(\mathbf{x}, \mathbf{z})$, given the current state. In the following demonstration, I and J are arbitrarily selected from $\{I_i\}_{i=1}^M$ and $\{J_j\}_{j=1}^N$.

The deterministic version of the algorithm computes updates of the form

$$\begin{aligned}\mathbf{x}_J^{k+1} &= \frac{1}{M}\sum_{i=1}^{M} {\mathbf{x}_J^i}^{k+1} \\ &= \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x}_J^k + \mu_j^i \mathbf{g}_J^i) \\ &= \mathbf{x}_J^k + \frac{1}{M}\sum_{i=1}^{M}\mu_j^i (A_{I_i}^J)^T(\mathbf{y}_{I_i} - \mathbf{z}_{I_i}^k),\end{aligned} \tag{16}$$

and

$$\begin{aligned}\mathbf{z}_I^{k+1} &= \sum_{j=1}^{N} {\mathbf{z}_I^j}^k = \sum_{j=1}^{N} A_I^{J_j}(\mathbf{x}_{J_j}^k + \mu_j^i \mathbf{g}_{J_j}^i) \\ &= \sum_{j=1}^{N} A_I^{J_j}(\mathbf{x}_{J_j}^k + \mu_j^i (A_I^{J_j})^T \mathbf{r}_I^k) \\ &= A_I\mathbf{x}^k + \sum_{j=1}^{N}\mu_j^i A_I^{J_j}(A_I^{J_j})^T(\mathbf{y}_I^k - \mathbf{z}_I^k) \\ &= A_I\mathbf{x}^k + S_I(\mathbf{y}_I^k - \mathbf{z}_I^k),\end{aligned} \tag{17}$$

where $S_I = \sum_{j=1}^N \mu_j^i A_I^{J_j}(A_I^{J_j})^T$.

This implies that, at the fixed point $\mathbf{x}^\star$ and $\mathbf{z}^\star$, we have

$$\mathbf{z}_I^\star = (I + S_I)^{-1} A_I \mathbf{x}^\star + (I + S_I)^{-1} S_I \mathbf{y}_I \tag{18}$$

and

$$\sum_{i=1}^{M}\mu_j^i (A_{I_i}^J)^T(\mathbf{y}_{I_i} - \mathbf{z}_{I_i}^\star) = \mathbf{0}. \tag{19}$$

Eq.18 can be expanded into the whole $\mathbf{z}$

$$\mathbf{z}^\star = (I + S_T)^{-1} A\mathbf{x}^\star + (I + S_T)^{-1} S_T \mathbf{y}, \tag{20}$$

where S_T is a block diagonal matrix:

$$S_T = \begin{bmatrix} S_{I_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & S_{I_2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & S_{I_M} \end{bmatrix} \tag{21}$$

Note that $(I + S_T)$ is a positive semi-definite matrix if the μ_j^i are positive. Define a diagonal matrix D_j with diagonal entries μ_j^i where $D_j(h,h) = \mu_j^i$ if $h \in I_i$. Eq.19 can thus be written as

$$(D_j A^J)^T(\mathbf{y} - \mathbf{z}^\star) = \mathbf{0} \tag{22}$$

Combine Eq.20 and Eq.22 gives

$$\begin{aligned}\mathbf{0} &= (D_j A^J)^T(\mathbf{y} - (I + S_T)^{-1} A\mathbf{x}^\star - (I + S_T)^{-1} S_T \mathbf{y}) \\ &= (D_j A^J)^T (I + S_T)^{-1}(\mathbf{y} - A\mathbf{x}^\star).\end{aligned} \tag{23}$$

This equation has to hold for all J. As D_j is a diagonal matrix, this implies that

$$\mathbf{0} = A^T(I + S_T)^{-1}(\mathbf{y} - A\mathbf{x}^\star), \tag{24}$$

which shows

$$\mathbf{x}^\star = \left(A^T(I + S_T)^{-1}A\right)^{-1} A^T(I + S_T)^{-1}\mathbf{y} \tag{25}$$

is a weighted least squares solution.

REFERENCES

[1] Y. Sagara, A. K. Hara, and W. Pavlicek, "Abdominal CT: comparison of low-dose CT with adaptive statistical iterative reconstruction and routine-dose CT with filtered back projection in 53 patients," *Am. J. Roentgenol.*, vol. 195, no. 3, pp. 713–719, Sep. 2010.

[2] E. J. Hoffman, S. C. Huang and M. E. Phelps, "Quantitation in positron emission computed tomography: 1. Effect of object size," *J. Comput. Assist. Tomo.*, vol. 3, no. 3, pp. 299–308, Jun. 1979.

[3] T. Rodet, F. Noo, and M. Defrise, "The cone-beam algorithm of feldkamp, davis, and kress preserves oblique line integrals," *Med. Phys.*, vol. 31, no. 7, pp. 1972–1975, Jun. 2004.

[4] L. Feldkamp, L. Davis and J. Kress, "Practical cone-beam algorithm," *JOSA A*, vol. 1, no. 6, pp. 612–619, Jun. 1984.

[5] A. Gervaise, B. Osemont, and S. Lecocq, "CT image quality improvement using adaptive iterative dose reduction with wide-volume acquisition on 320-detector CT," *Euro. Radio.*, vol. 22, no. 2, pp. 295–301, Feb. 2012.

[6] G. Wang, H. Yu, and B. De Man, "An outlook on x-ray ct research and development," *Med. Phys.*, vol. 35, no. 3, pp. 1051–1064, Feb. 2008.

[7] J. Deng, H. Yu, and J. Ni, "Parallelism of iterative ct reconstruction based on local reconstruction algorithm," *J. Supercomput.*, vol. 48, no. 1, pp. 1–14, Apr. 2009.

[8] M. J. Willemink, P. A. Jong and T. Leiner, "Iterative reconstruction techniques for computed tomography Part 1: technical principles," *Eur. Radiol*, vol. 23, no. 6, pp. 1623–1631, Jun. 2013.

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

[10] X. Guo, "Convergence studies on block iterative algorithms for image reconstruction," *Appl. Math. Comput.*, vol. 273, pp. 525–534, Jan. 2016.

[11] M. Beister, D. Kolditz, and W. A. Kalender, "Iterative reconstruction methods in x-ray CT," *Phys. Medica*, vol. 28, no. 2, pp. 94–108, Apr. 2012.

[12] F. Jacobs, E. Sundermann, B. De Sutter, and M. Christiaens, "A fast algorithm to calculate the exact radiological path through a pixel or voxel space," *J. CIT.*, vol. 6, no. 1, pp. 89–94, Mar. 1998.

[13] J. Ni, X. Li, T. He, and G. Wang, "Review of parallel computing techniques for computed tomography image reconstruction," *Curr. Med. Imaging Rev.*, vol. 2, no. 4, pp. 405–414, Nov. 2006.

[14] M. Zibetti, C. Lin and G. Herman,"Total variation superiorized conjugate gradient method for image reconstruction,"*Inver. Prob.* vol. 34, no. 3, pp. 1–26, Jan. 2018.

[15] M. Chillaron, V. Vidal, D. Segrelles, I. Blanquer and G. Verdu,"Combining grid computing and docker containers for the study and parametrization of CT image reconstruction methods,"*Procedia Comput. Sci.* vol. 108, pp. 1195–1204, Jun. 2017.

[16] S. Karimi, F. Toutounian, "The block least squares method for solving nonsymmetric linear systems with multiple right-hand sides,"*Appl. Math. Comput.* vol. 177, no. 2, pp. 852–862, Jun. 2006.

[17] P. Dianne, O. Leary, "The block conjugate gradient algorithm and related methods,"*Linear Algebra Appl.* vol. 29, pp. 293–322, Feb. 1980.

[18] J. Hsieh, B. Nett, and Z. Yu, "Recent advances in CT image reconstruction," *Cur. Radio. Repor.*, vol. 1, no. 1, pp. 39–51, Mar. 2013.

[19] J. Bilbao-Castro, J. Carazo, J. Fernández, and I. Garcia, "Performance of parallel 3D iterative reconstruction algorithms," *WSEAS Trans. on Bio. and Biomedi.*, vol. 1, no. 1, pp. 112–119, Jan. 2004.

[20] Y. Censor, "Row-action methods for huge and sparse systems and their applications," *SIAM Rev.*, vol. 23, no. 4, pp. 444–466, Jun. 1981.

[21] D. W. Watt, "Column-relaxed algebraic reconstruction algorithm for tomography with noisy data," *Appl. Opt.*, vol. 33, no. 20, pp. 4420–4427, Sep. 1994.

[22] T. Elfving, "Block-iterative methods for consistent and inconsistent linear equations," *Numer. Math.*, vol. 35, no. 1, pp. 1–12, Mar. 1980.

[23] J. Kole and F. Beekman, "Parallel statistical image reconstruction for cone-beam x-ray CT on a shared memory computation platform," *Phys. Med. Biol.*, vol. 50, no. 6, pp. 1265–1272, Mar. 2005.

[24] T. Li, T. J. Kao, and Isaacson, "Adaptive kaczmarz method for image reconstruction in electrical impedance tomography," *Physiol. Meas.*, vol. 34, no. 6, p. 595, May. 2013.

[25] Y. Censor, P. P. Eggermont, and D. Gordon, "Strong underrelaxation in kaczmarz's method for inconsistent systems," *Numer. Math.*, vol. 41, no. 1, pp. 83–92, Feb. 1983.

[26] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm," *Ultra. Imag.*, vol. 6, no. 1, pp. 81–94, Jan. 1984.

[27] Y. Censor, "Parallel application of block-iterative methods in medical imaging and radiation therapy," *Math. Program.*, vol. 42, no. 1, pp. 307–325, Apr. 1988.

[28] J. Gregor and T. Benson, "Computational analysis and improvement of SIRT," *IEEE Trans. on Med. Ima.*, vol. 27, no. 7, pp. 918–924, Jun. 2008.

[29] N. D. Tang, N. De Ruiter, J. Mohr, and A. P. Butler, "Using algebraic reconstruction in computed tomography," in *27th Conf. IVCNZ, Dunedin, New Zealand*, pp. 216–221, ACM, Nov. 2012.

[30] T. M. Benson and J. Gregor, "Framework for iterative cone-beam micro-CT reconstruction," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 5, pp. 1335–1340, Oct. 2005.

[31] H. M. Hudson and R. S. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," *IEEE Trans. Med. Imag.*, vol. 13, no. 4, pp. 601–609, Dec. 1994.

[32] Y. Censor, D. Gordon, and R. Gordon, "Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems," *Parallel Comput.*, vol. 27, no. 5, pp. 777–808, May. 2001.

[33] Y. Censor and T. Elfving, "Block-iterative algorithms with diagonally scaled oblique projections for the linear feasibility problem," *SIAM. J. Matrix Anal. and Appl.*, vol. 24, no. 1, pp. 40–58, Jul. 2002.

[34] Y. Censor, D. Gordon, and R. Gordon, "BICAV: A block-iterative parallel algorithm for sparse systems with pixel-related weighting," *IEEE Trans. Med. Imag.*, vol. 20, no. 10, pp. 1050–1060, Oct. 2001.

[35] K. Sauer and C. Bouman, "A local update strategy for iterative reconstruction from projections,"*IEEE Trans. Signal Process*, vol. 41, no. 2, pp. 534–548, Feb. 1993.

[36] C. Bouman and K. Sauer, "A unified approach to statistical tomography using coordinate descent optimization,"*IEEE. Trans. Signal Process*, vol. 5, no. 3, pp. 480–492, Mar. 1996.

[37] J. Thibault, K. Sauer, C. Bouman and J. Hsieh, "A threedimensional statistical approach to improved image quality for multislice helical CT,"*Med. Phys.*, vol. 34, no. 11, pp. 4526–4544, Oct. 2007.

[38] T. M. Benson, B. K. De Man, L. Fu, and J.-B. Thibault, "Block-based iterative coordinate descent," in *NSS/MIC,2010 IEEE*, pp. 2856–2859, IEEE, 2010.

[39] D. Kim and J. 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*, pp. 33–36, 2012.

[40] K. Sauer, S. Borman and C. Bouman, "Parallel computation of sequential pixel updates in statistical tomographic reconstruction,"*IEEE. ICIP*, Oct. 1995.

[41] J. Fessler, "Grouped-coordinate ascent algorithms for penalized-likelihood transmission image reconstruction,"*IEEE. Trans. Med. Ima.*, vol. 16, no. 2, pp. 166–175, Apr. 1997.

[42] J. Zhang, S. Saquib and K. Sauer, "Parallelizable Bayesian tomography algorithms with rapid, guaranteed convergence,"*IEEE. Trans. Ima. Pro.*, vol. 9, no. 10, pp. 1745–1759, Oct. 2000.

[43] J. Fessler and D. Kim,"Axial block coordinate descent (ABCD) algorithm for x-ray CT image reconstruction,"*Proc. Fully Three-Dimensional Image Reconstruct. Radiol. Nucl. Med.*, Jul. 2011.

[44] C. Hsieh, K. Chang and C. Lin, "A Dual Coordinate Descent Method for Large-scale Linear SVM,"*ICML.* pp. 408–415, Jul. 2008.

[45] K. Chang, C. Hsieh and C. Lin, "Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines,"*J. Mach. Learn. Res.* vol. 9, no. 2, pp. 1369–1398, Jul. 2008.

[46] Z. Yu, J. B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh, "Non-homogeneous updates for the iterative coordinate descent algorithm," in *Electronic Imaging 2007, San Jose, CA, United States*, pp. 64981B1–64981B12, International Society for Optics and Photonics, Feb. 2007.

[47] Z. Yu, J.-B. Thibault, C. A. Bouman, and K. D. Sauer, "Fast model-based x-ray CT reconstruction using spatially nonhomogeneous icd optimization," *IEEE Trans. Image Process*, vol. 20, no. 1, pp. 161–175, Jan. 2011.

[48] W. J. Palenstijn, J. Bédorf, and K. J. Batenburg, "A distributed SIRT implementation for the ASTRA toolbox," in *Proc. Fully Three-Dimensional Image Reconstruct. Radiol. Nucl. Med.*, pp. 166–169, Jun. 2015.

[49] J. Chen and Q. Gu, "Accelerated Stochastic Block Coordinate Gradient Descent for Sparsity Constrained Nonconvex Optimization,"*UAI Proc.* pp. 132–141, Jun. 2016.

[50] A. Zhang and Q. Gu, "Accelerated Stochastic Block Coordinate Descent with Optimal Sampling,"*Proc. of the ACM SIGKDD Inter. Conf.* pp. 2035–2044, Aug. 2016.

[51] H. Wang and A. Banerjee, "Randomized block coordinate descent for online and stochastic optimization,"*arXiv preprint arXiv:1407.0107* Jul. 2014.

[52] T. Zhao, M. Yu, Y. Wang, R. Arora and H. Liu, "Accelerated Mini-batch Randomized Block Coordinate Descent Method,"*NIPS* pp. 3329–3337, Dec. 2014.

[53] R. Johnson and T.Zhang, "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction,"*NIPS* pp. 315–323, Dec. 2013.

[54] J. Konecny, Q. Zheng and P. Richtarik, "Semi-stochastic coordinate descent,"*Optim. Methods Sofw.* vol. 32, no. 5, pp. 993–1005, Mar. 2017.

[55] Y. Xu and W. Yin,"Block Stochastic Gradient Iteration for Convex and Nonconvex Optimization,"*SIAM J. Optim* vol. 25, no. 3, pp. 1686–1716, Aug. 2015.

[56] Z. Lu and L. Xiao, "On the complexity analysis of randomized block-coordinate descent methods,"*Math. Program* vol. 152, no. 1, pp. 615–642, Aug. 2015.

[57] M. Neumann and R. Plemmons, "Convergence of parallel multisplitting iterative methods for M-matrices,"*Numer. Linear Algebra Appl.* vol. 88, pp. 559–573, Apr. 1987.

[58] R. Renaut, "A parallel multisplitting solution of the least squares problem,"*Numer. Linear Algebra Appl.* vol. 5, no. 1, pp. 11–31, May, 1998

[59] A. Frommer, G. Mayer, "Convergence of relaxed parallel multisplitting methods,"*Linear Algebra Appl* vol. 119, pp. 141–152, Jul. 1989.

[60] R. Wen and H. Duan, "A parallel multisplitting method with self-adaptive weightings for solving H-matrix linear systems,"*J. Inequal. Appl.* vol. 2017, no. 1, pp. 95–105, Jun. 2017.

[61] S. Boyd, N. Parikh, E. Chu, B. P and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers,"*Foundations and Trends in Machine Learning* vol. 3, no. 1, pp. 1–122, Jul. 2011.

[62] W. Deng, M. Lai, Z. Peng and W. Yin, "Parallel multi-block ADMM with *o* (1/k) convergence,"*J. Sci. Comput.* vol. 71, no. 2, pp. 712–736, May. 2017.

[63] N. Parikh and S. Boyd, "Block splitting for distributed optimization,"*Mathm. Prog. Comput.* vol. 6, no. 1, pp. 77–102, Mar. 2014.

[64] W. Aarle, W. Palenstijn, J. Beenhouwer, T. Altantzis, S. Bals, K. Batenburg, J. Sijbers, "The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography,"*Ultramicroscopy*vol. 157, pp. 35–47, May 2015.

[65] A. Biguri, M. Dosanjh, S. Hancock, and M. Soleimani,"TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction,"*Biomed. Phys. Eng. Express* vol. 2, no. 5, pp. 055010, Sep. 2016.

[66] J. Langford, A.J. Smola and M. Zinkevich, "Slow learners are fast,"*Advan. in Neu. Info. Proc. Sys.* pp. 2331–2339, 2009.

[67] A. Alekh and J.C. Duchi,"Distributed delayed stochastic optimization,"*Advan. in Neu. Info. Proc. Sys.* pp. 873–881, 2011.