UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING AND PHYSICAL SCIENCE

Institute of Sound and Vibration Research

Fast, large scale optimization algorithms for tomographic image reconstruction

by

Yushan Gao

Supervisor: Thomas Blumensath, Paul White

Examiner: Jordan Cheer, Bill Lionheart

March 17, 2021

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCE
Institute of Sound and Vibration Research

FAST, LARGE SCALE OPTIMIZATION ALGORITHMS FOR TOMOGRAPHIC IMAGE RECONSTRUCTION

by Yushan Gao

Tomography imaging techniques produce volumetric images of the three-dimensional structure of an object. X-ray radiation is one of the standard modalities used for three-dimensional imaging and in this case, X-ray projection images are typically collected from the object at different orientations. These projections are then used to compute a volumetric representation of the object's internal x-ray attenuation profile. Scientific and industrial tomographic imaging applications require the use of ever more massive datasets as they increasingly use larger and higher resolution detectors and use increasing numbers of projections to scan the object with the required resolution. Furthermore, the need to discern ever-finer details within an object leads to an increase in the desired resolution of the reconstructed volume. If this is paired with the usage of non-standard tomographic scanning trajectories, then the filtered back-projection algorithm, which remains the primary workhorse for the tomographic reconstruction of large datasets, is no longer applicable. Compared to back-projection based methods, iterative algorithms have many advantages for linear tomographic image reconstruction. However, for large-scale tomographic reconstruction using computation nodes with limited storage capacity, the projection data and the reconstructed image vector have to be both partitioned into many smaller blocks. Each iteration in a traditional iterative method needs access to either all projection data or to the entire image (or to both) and thus needs to iterate over individual blocks that need to be copied to the processing node. This additional data access can significantly reduce reconstruction speed. To address these challenges, this project develops novel algorithms that are tailored to large-scale tomographic reconstruction. The algorithms are designed to fit on modern high-performance computing infrastructures, where each computation node does not have fast access to the entire dataset at once and where communication between different nodes is relatively slow. This thesis includes the introduction of the developed algorithms, the comparison of them with existing methods and the application of them on realistic parallel network.

Declaration of Authorship

I, Yushan Gao , declare that the thesis entitled *Fast, large scale optimization algorithms for tomographic image reconstruction* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

Signed:.....

Date:.....

Acknowledgements

I would like to thank my supervisors, especially Dr Thomas Blumensath, for his support during four years' studies at the University of Southampton. During four-year research, every time when my research meets challenges, my supervisor is always ready to provide insightful opinions and ideas. Every time when I submit a report, Thomas always provide detailed feedbacks and revise suggestions. Every time when I prepare a paper, Thomas always help me to improve it as much as he can, even including helping me correct grammar mistakes, which is far beyond his supposed duty. Thomas not only teaches me project-related knowledge, but also influences me on wider aspects. His rigorous attitude towards scientific research and efficient working habits leave me a very deep impression and I am determined to learn from him in my future career.

I also would like to thank researcher Dr Boardman Richard in $\mu - vis$ laboratory and my colleagues Josh Greenhalgh, Dr Ander Biguri and Lindroos Reuben, who are always ready to help me on any computer-related difficulties I encountered. Especially at my final stage when I need to run TIGRE toolbox on Iridis 5 clusters, Ander and Lindroos helped me a lot in terms of the installation issue even on weekends. Furthermore, many thanks for my second supervisor Prof. Paul White and Dr Jordan Cheer as being my examiners in my eighteen-month's viva. Fruitful suggestions have been proposed and these feedbacks play important roles in my remained research activities. Many thanks to you and wish you will always enjoy your lives in future.

Finally, I would like to thank my best friend Fanzi Liu. You are the person I admire the most through all peers I met in UK. Thank you for your positive attitude towards life and your tough spirit that always encourage me to face various challenges. You really set a good example for me and there are so many things that I should learn from you. Also thank you for your music which accompanies me through lonely working nights and through countless ups and downs.

The study life in University of Southampton is my happiest time in my past 28 years and I am so lucky to meet so many good people: a fantastic supervisor, kind and helpful colleagues and my life-long friends. With all your generous help, I believe that I am always on the way to becoming a better man, both in academic research domain and in daily life. I will never say goodbye to you and to the university!

Contents

List of Figures

List of Tables

List of Symbols

Mathematics

$\mathbb{R}^{r*c}$	real matrix with r rows c columns set
I_{in}	input X-ray intensity
I_{out}	output X-ray intensity
a	thickness of an object with the same attenuation coefficient
x	attenuation coefficient of an uniform object
a_x	the axil of the horizontal direction for a two dimension image
a_y	the axil of the vertical direction for a two dimension image
$\mathbf{A}$	system matrix
r	number of rows of $\mathbf{A}$
c	number of columns of $\mathbf{A}$
$\mathbf{y}$	projection data
$\mathbf{e}$	noise vector blurring projection data
$\mathbf{x}_{true}$	scanned image vector
$\mathbf{x}_{rec}$	reconstructed image vector during reconstructions
$a_{i,j}$	element of $\mathbf{A}$ of i^{th} row and j^{th} column
M	the number of row blocks
N	the number of column blocks
I_i	indexes of i^{th} row block
J_j	indexes of j^{th} column block
$\hat{J}$	complement set of J_j
I	index of a random row block
J	index of a random column block
$\mathbf{x}_{J_j}$	a block of $\mathbf{x}$ indexed by J_j
$\mathbf{y}_{I_i}$	a block of $\mathbf{y}$ indexed by I_i
$\mathbf{A}_{I_i}^{J_j}$	a sub-matrix of $\mathbf{A}$
m	number of rows of $\mathbf{A}_{I_i}^{J_j}$
n	number of columns of $\mathbf{A}_{I_i}^{J_j}$
$\mathbf{A}_i$	i^{th} row of A
$\mathbf{R}, \mathbf{S}, \mathbf{D}, \mathbf{C}$	weight matrices defined in various iterative algorithms
$\tilde{R}, \tilde{C}$	diagonal elements of weight matrix $\mathbf{R}, \mathbf{C}$

$\mathbf{g}$	iterative direction in iterative algorithms, it can be a full gradient or a stochastic gradient of the objective function
$\mathbf{g}_{true}^{k}$	full gradient direction at k^{th} epoch
$\mathbf{g}_{i}$	a stochastic estimation of gradient $\mathbf{g}_{true}$ using the i^{th} row of $\mathbf{A}$
μ	step-length in iterative algorithms
$\mathbf{e}_{j}$	j^{th} vector in standard basis
$\mathbf{z}_{I_i}^{j}$	stores the forward projection results of $\mathbf{x}_{J_j}$ using sub matrix $\mathbf{A}_{I_i}^{J_j}$
$\nabla\mathbf{z}_{I_i}^{j}$	difference of two adjacent $\mathbf{z}_{I_i}^{j}$ during iterations
$\hat{\mathbf{g}}_{J_j}^{i}$	stores the back projection results of $\mathbf{r}_{I_i}$ using sub matrix $(\mathbf{A}_{I_i}^{J_j})^T$
$\nabla\hat{\mathbf{g}}_{J_j}^{i}$	difference of two adjacent $\hat{\mathbf{g}}_{J_j}^{i}$ during iterations
α	the fraction of selected row blocks
γ	the fraction of selected column blocks
β	relaxation parameter in CSGD
λ	regularization parameter in total variation(TV) discussions
K_{max}	the maximum number of loops in iterative algorithms
$\mathbf{x}_{solution}$	the final reconstructed image vector
$\mathbf{x}_{lsq}$	least square solution when dataset is blurred by noises
k	epoch numbers in iterative algorithms
K	pixel and voxel number of the reconstructed image on one physical dimension
p	total number of parallel nodes in a parallel network
σ	noise variance
OP	distance of X-ray point to rotation center in circular scanning geometry
OD	distance of object center to rotation center in circular scanning geometry
$\tilde{r}$	the size that projection data takes in a computer
$\tilde{c}$	the size that reconstruction volume takes in a computer
G	storage limit of a computation node
$\mathbf{x}^{\star}$	a fixed point of iteration process. i.e. $\mathbf{x}^{\star} = M\mathbf{x}^{\star} + \mathbf{c}$, where M is an iteration matrix and $\mathbf{c}$ is a vector
θ	projection view
t	virtual time which is used to simulate the actual wall time in asynchronous communication environment when using serial codes
l	parallel node index
$t_{commu,l}$	virtual time spent on communication between l^{th} node and the master node
$t_{proj,l}$	virtual time spent on FP and BP in l^{th} node
t_{upd}	virtual time spent on updating procedure

$\tau_{commu,l}$	parameters influencing $t_{commu,l}$
$\tau_{proj,l}$	parameters influencing $t_{proj,l}$
τ_{upd}	parameters influencing t_{upd}
$\tilde{t}_l$	predicted virtual wall time for l^{th} node when it finishes current computation
DS	l_2 norm of the difference of $\mathbf{x}_{rec}$ and the least square solution $\mathbf{x}_{lsq}$
RE	relative error of calculated $\mathbf{x}$ and $\mathbf{x}_{true}$
$\{J_j\}_{j=1}^{N}$	an index set including $J_1, \cdots, J_N$. It is also expressed as $\{J_j\}_{j\in[1,N]}$

Operators

$\nabla f(\mathbf{x})$	gradient of $f(\mathbf{x})$
$f(\mathbf{x})$	objective function defined as $\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2$
$g(\mathbf{x})$	regularization function for $\mathbf{x}$
$TV(\mathbf{x})$	calculate the total variation value of image $\mathbf{x}$
$\| \bullet \|$	l_2 norm of a vector or matrix
$f_{I_i}(\mathbf{x})$	defined as $\frac{1}{2}\|\mathbf{y}_{I_i} - \mathbf{A}_{I_i}\mathbf{x}\|^2$
$\mathbb{E}(\bullet)$	expectation of a matrix
$\binom{Q}{P}$	random select P elements from Q-elements set
$diag(a_1, \cdots, a_n)$	create a diagonal matrix whose diagonal elements are $a_1, \cdots, a_n$
$[1, M]$	a set that contains elements $1, \cdots, M$ $(M > 1)$

List of Acronyms

2D	Two Dimension, first defined in p1
3D	Three Dimension, p2
ADMM	Alternating Direction Method of Multipliers, p9
ART	Algebraic Reconstruction Technology, p7
ASD-POCS	Adaptive Steepest Descent Projection Onto Convex Sets, p38
BICAV	Block Iterative Component Averaging Method, p25
BP	Backward projection, p7
BSGD	Block Stochastic Gradient Descent, p9
BSGD-TV	BSGD combined with TV denoising procedure, p103
BSGD-IM	BSGD accelerated by Importance sampling, p82
CAV	Component Averaging Method, p7
CD	Coordinate Descent, p28
CG	Conjugate Gradient, p43
CGLS	Conjugate Gradient for the Least Square problems, p43
CPU	Central Processing Unit, p8
CSGD	Coordinate-reduced Steepest Gradient Descent, p9
CSGD-TV	CSGD combined with TV regularizations, p103
CSGD-IM	CSGD accelerated by Importance sampling, p69
CS	Compressed Sensing, p36
CT	Computed Tomography, p1
EASGD	Elastic Averaging Stochastic Gradient Descent, p31
FBP	Filtered Back-projection, p3
FDK	Feldkamp Davis Kress, p3
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm, p35
FP	Forward projection, p7
FT	Fourier transform, p4
GD	Gradient Descent, p19
GPU	Graphics Processing Unit, p8
IFT	Inverse fourier transform, p4
KKT	Karush–Kuhn–Tucker conditions, p93
MPI	Message Passing Interface, p118
RAM	Random Access Memory, p76

RBCD	Randomised Block Coordinate Descent, p29
SAG	Stochastic Averaging Gradient, p8
SART	Simultaneous Algebraic Reconstruction Technique, p25
SBCD	Stochastic Block Coordinate Descent, p30
SF matrix	Short-fat matrix, p61
SGD	Stochastic Gradient Descent, p8
SIRT	Simultaneous Iterative Reconstruction Technique, p7
SNR	Signal-to-noise Ratio, p4
SVRG	Stochastic Variance Reduced Gradient, p8
TT matrix	Tall-thin matrix, p63
TV	Total Variation, p8

Chapter 1

Introduction

1.1 Thesis background

Computed tomography (CT) is a non-destructive and non-invasive technique to produce the internal structure of an object. It uses X-rays, gamma rays, etc. as the imaging medium. The light source and detector scan the target along a certain trajectory to obtain projections of a certain physical quantity of the object. After computer processing projections, the distribution of the certain parameter is then obtained. Since the 1970s, CT has become a basic imaging method in the research of biomedicine and industrial testing (Bayford and Lionheart, 2004; Bartscher et al., 2007; Taina et al., 2008; Schindler et al., 2017; De Chiffre et al., 2014; Gholizadeh, 2016).

To derive the linear model that will be at the heart of this thesis, we consider a single-energy X-ray beam, which contains N_{in} monochromatic photons that pass through a uniform object of length a. Due to the photoelectric effect and the Compton effect, the rays passing through the object are attenuated and scattered. The number of photons is thus reduced to N_{out}. According to Lamber-Beer law (Soleimani and Pengpen, 2015):

$$N_{out} = N_{in}e^{-xa} \tag{1.1}$$

or

$$I_{out} = I_{in}e^{-xa}, \tag{1.2}$$

where I_{in} and I_{out} are incident and detected X-ray intensity. x is the attenuation coefficient within the homogeneous object.

When X-rays penetrate an inhomogeneous object, x becomes a function of two spacial coordinate, a_x and a_y, $x(a_x, a_y)$, as shown in Fig.1.1. The relationship between I_{out} and I_{in} along a certain path L is then

$$I_{out} = I_{in}e^{-\int_L x(a_x,a_y)dl}. \tag{1.3}$$

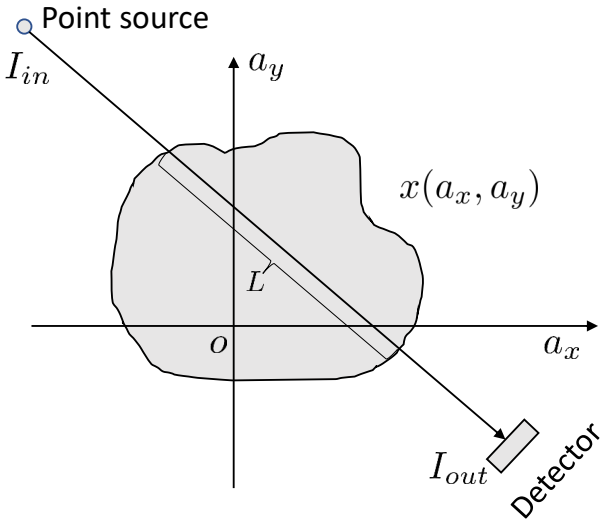


Figure 1.1: Diagram showing the X-ray penetrating an object with non-homogeneous attenuation coefficients.

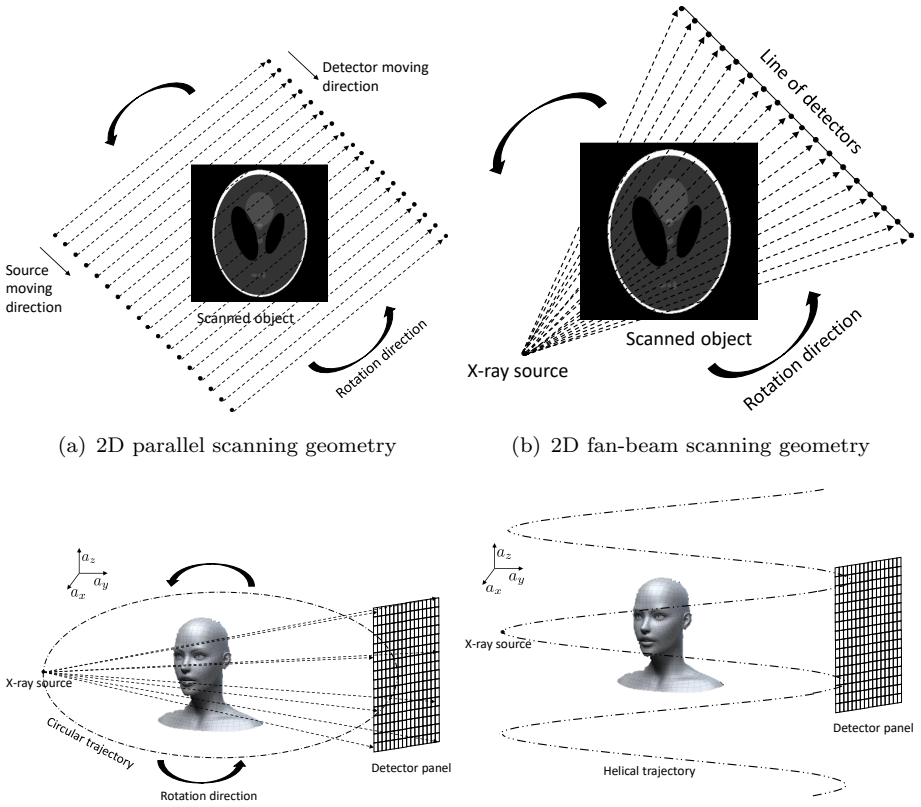
or equivalently,

$$y_i = ln\left(\frac{I_{in}}{I_{out}}\right) = \int_L x(a_x, a_y)dl. \tag{1.4}$$

In CT scanning, measurements y_i are taken over many different paths L. Common scan trajectories include: parallel-beam, fan-beam, cone-beam and helical scan. In parallel beam scanning, measurements are taken along parallel beams, either in a 2D plane or in a 3D volume (see Fig.1.2(a)). The object or the source and detector are then rotated and repeated parallel paths are measured for different rotation angles. Parallel beam CT is common at synchrotron imaging beamlines, where x-ray paths are approximately parallel once they reach the object. Fan-beam scanning uses a point like x-ray source and a linear detector. As the source is relatively close to the object, at each rotation angle, measured X-ray paths follow a fan shape. Again, the object or the source/detector pair are then rotated and several projection measurements are taken form different directions. Cone-beam scanning is an extension of fan-beam from 2D to 3D. A 2D detector panel is used to collect projection images using a point source. The object or the source and detector are again rotated and measurements taken from a range of angles. Helical scanning is similar to cone-beam scanning, but now the object or the source and detector pair also move at a constant rate in the direction of the rotation axis. The scanning geometry is illustrated in Fig.1.2. In this thesis, the fan and cone-beam scan geometries are used throughout unless stated otherwise.

1.2 CT reconstruction algorithms

Given a set of measurements y_i, the CT reconstruction problem is the estimation of the x-ray attenuation $x(a_x, a_y)$. For most X-ray CT scan trajectories, this is known to be an



(a) 2D parallel scanning geometry (b) 2D fan-beam scanning geometry

(c) 3D cone-beam circular scanning geometry (d) 3D cone-beam helical scanning geometry

Figure 1.2: Popular scanning geometries in 2D and 3D CT.(a) shows a parallel scanning with single source-detector pair. (b) shows a equal spaced fan-beam scanning geometry whereas equal angular fan-beam scanning is omitted here. (c) shows a cone-beam circular scanning model for 3D volume. The point source and the central line of the detector plane locate at the middle of the volume. (d) shows a cone beam helical scanning trajectory. In the following presentation, fan-beam(b) and cone-beam(circular trajectory, i.e.(c)) scanning geometries are default settings in 2D and 3D simulations.

ill-posed inverse problem. For many scan trajectories, approximate analytical methods exist solutions, though many, more general, iterative algorithms have also been proposed. Among the analytical methods, the filtered back projection (FBP) algorithm is currently the most commonly used method for parallel-beam and fan-beam scans. (Gordon et al., 1975; Turbell, 2001). The algorithm is easy to be implemented in hardware and can compute good image estimates when sufficiently many measurements have been taken. In 3D cone-beam reconstruction, the Feldkamp Davis Kress method (FDK), which is an approximate application of FBP to 3D case, has been widely used in commercial CT machines (Hsieh et al., 2013).

Iterative methods start with a discretisation of the forward model and then apply iterative optimisation algorithms to estimate the x-ray attenuation (Censor, 1983). Iterative methods have the advantage that they work for generic trajectories. They also often provide better results when limited measurements are available. Importantly, iterative

methods can often be easily adapted to use a variety of constraints to stabilize the reconstructed image when the dataset has limited projections and when measurement noise is amplified due to the ill-conditioning of the model (Censor, 1983).

1.2.1 Analytical methods

The FBP is proposed for 2D (and slice wise reconstruction of 3D) parallel beams and fan-beam reconstructions based on the central slice theorem (Gordon et al., 1975; Turbell, 2001). The reconstruction process is mainly divided into two steps: the first step is to perform Fourier transform(FT) on the projection data and a filter is multiplied on the FT results. Then an inverse FT (IFT) is applied on the filtered results. In the second step, the IFT results is back-projected and accumulated to obtain the final image. In the actual calculation, the bandwidth of the filter applied in the first step must be limited, so there are various smooth truncation windows to relieve the aliasing artifacts caused by the filter's limited bandwidth , such as Shepp-logan filter, Cosine filter, Hamming filter, Hann filter (Kak, 1979; Othman et al., 2016; Hu, 1999; Hsieh, 2003). The filters effectively regularise the reconstruction.

The FDK algorithm is an approximate reconstruction algorithm for cone-beam circular scanning. (Manzke et al., 2005; Hsieh et al., 2013). It is actually an approximation of a 3D extension of the FBP algorithm. The algorithm regards cone-beam rays as a combination of fan-beam rays with different angles in the a_z-axis (defined in Fig.1.2(c) and Fig.1.2(d)) direction. Accurate FBP is achieved on the central plane. For non-central planes, the fan-beam reconstruction formula is modified to perform approximate reconstruction. When the cone angle in a_z direction is small, the reconstruction result is good, but since it is an approximate algorithm, when the cone angle becomes large, the reconstruction quality degrades severely. Another fact of FDK is that it is only applied in the case when the scanning trajectory is circular, whose curve theoretically does not meet Tuy's condition for accurate reconstruction (Natterer, 2001). As a result, for the case where high-precision reconstruction is required, FDK and circular scanning methods often cannot meet the requirements. Precise reconstruction algorithms for helical scanning trajectory, which meet Tuy's condition, have been proposed, such as Grangeat algorithm (Grangeat, 1991), Katsevich algorithm (Katsevich, 2002a,b), BPF algorithm (Zou and Pan, 2004), etc. However, the discussion of them is beyond the scope of the thesis and thus is omitted here.

FBP and FDK have been widely used in the field of medical image reconstruction because of their computational efficiencies and numerical stabilities. Generally, analytical methods directly calculate the reconstructed image without an iterative process and thus a visually acceptable reconstruction result can be quickly obtained whenever there is sufficient projection data and the projection noise is not too large. However, the main disadvantage of analytical methods is that they require equally spaced projections

and high signal-to-noise ratio (SNR) projection data (Wang et al., 2008; Flores et al., 2014). The implicit regularisation used is also hard to control as filtering is done in the projection domain. To obtain uniformly spaced projections, analytical methods have to demand the scanning trajectory to be strictly circular or helical. For medical applications, this is easy to achieve if imaging relatively small and nearly cylindrical objects. However, for many industrial applications, sometimes the object is too large to allow the X-ray source to rotate around it, or objects are too dense for X-ray penetration in certain directions. In these cases, only limited views or even random views are available and significantly stronger regularization terms are needed to stabilise the solution.

1.2.2 Iterative methods

When the projection data is limited or the projection angle is not uniformly distributed, strong regularization plays an important role in stabilising the reconstruction. (This will be introduced in the next chapter.) The analytical methods then often cannot get results of sufficient quality. Iterative methods, which can be combined easily with explicit regularization terms, can overcome the shortcomings of analytical methods, and can obtain higher-quality images in this case. Different from the analytical methods, the iterative methods discretize the image from the very beginning. The fundamental model to the 2D X-ray CT reconstruction is formulated in the following way: A cartesian grid of square picture elements, called pixels, is introduced into the space of scanned object so that it covers the whole object. The pixels are numbered in some agreed manner, say from 1(top left corner) to c(bottom right corner pixel), as shown in Fig.1.3. The model of 3D scanning is similar to the 2D case by expanding pixels into cubic cell, called voxels. The X-ray attenuation function is assumed to take a constant value x_j throughout the j_{th} pixel/voxel for $j = 1, 2, \cdots, c$. The source and detector are still both assumed to be points and the rays between them to be lines. Further, assume that the length of intersection of the i_{th} ray with the j_{th} pixel/voxel, denoted by a_{ij} for all $i = 1, 2, \cdots, r, j = 1, 2, \cdots, c$, represents the weight of the contribution of the j_{th} pixel/voxel to the total attenuation along the i_{th} ray. The physical measurement of y_i (defined in Eq.1.4), which represents the line integral of the unknown attenuation function along the path of the ray, turns out to be a finite sum in this discretized model:

$$y_i = \sum_{j=1}^{c} a_{ij} x_j + e_i, i = 1, 2, \cdots, r. \tag{1.5}$$

In matrix notation we write Eq.1.5 as

$$\mathbf{y} = \mathbf{A}\mathbf{x}_{true} + \mathbf{e}, \tag{1.6}$$

where $\mathbf{A} \in \mathbb{R}^{r \times c}$ is the system matrix and $\mathbf{e}$ is the error vector, which represents the measurement inaccuracy, noise corruption of data and the fact that the original problem

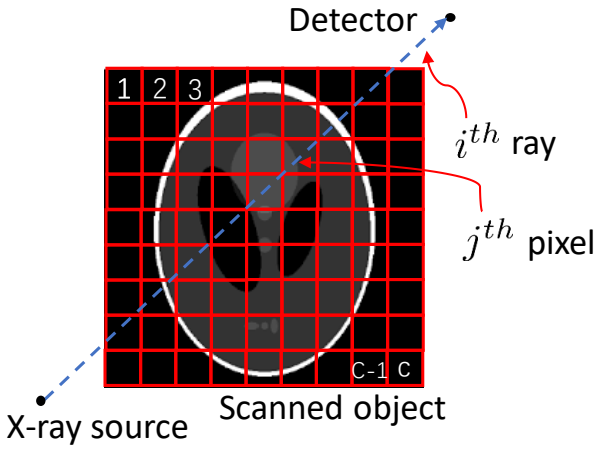


Figure 1.3: The square pixels model for 2D tomography image reconstruction. Within the pixel the attenuation x is assumed to be constant. It can be seen that if the size of pixels is too large, then some pixels contain various attenuation information inside, which means that the discretization of the continuous image causes error and this error decreases as the size of the pixel decreases. For simplicity, in this thesis, the error caused by discretization is not considered.

has undergone discretization. One common source of the error vector comes from the fact that X-ray interactions follow a Poisson stochastic process. According to the definition of y_i in Eq.1.4, randomness in the the measurement of y_i is introduced by statistical fluctuations in N_{out} whilst N_{in} may considered to be known with negligible error. A detailed discussion on the error is beyond the scope of the thesis. In general it can be concluded that when the number of N_{out} is large enough, it is safe to assume that $\mathbf{e}$ is approximately Gaussian. As a result, in the following of the thesis, the noise contaminating the projection data $\mathbf{y}$ is assumed to be independent and identically distributed Gaussian noise.

The maximum likelihood estimation in iterative methods then becomes solving the least squares cost function $\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2$. Many iterative algorithms which will be presented in next chapter can be understood as iterative optimisation algorithms that try to minimise this cost function, or to minimise a cost function that is similar to this one. Detailed discussions can be found in the next chapter. It needs to be noted that due to ill-conditioning of the system, minimising the cost function does not guarantee small reconstruction errors measured by the signal-to-noise ratio (SNR) of the reconstructed image. The SNR is defined as

$$\text{SNR} = 20log_{10}\frac{\|\mathbf{x}_{true}\|_2}{\|\mathbf{x}_{rec} - \mathbf{x}_{true}\|_2}, \tag{1.7}$$

where $\mathbf{x}_{rec}$ and $\mathbf{x}_{true}$ are reconstructed vector and the original discretized true vector respectively. In this thesis, the SNR reflects the reconstruction quality during solving the linear system. In CT reconstruction, it is important to realise that(Censor, 1983): 1)The matrix $\mathbf{A}$ is very sparse with less than 1 percent of its entries non-zero because

only few pixels have a non-zero intersection with each X-ray. 2)The system matrix $\mathbf{A}$ is extremely large, with r and c of the order of magnitude 10^5 or even higher, in order to produce images with good resolution. Although the matrix is sparse, the storage of $\mathbf{A}$ is still often infeasible due to the enormous number of non-zeros. 3) The linear system is typically severely ill posed and sometimes under-determined due to lack of projection information. In 2D parallel or fan-beam scanning geometry, the system can be"mildly ill-posed". However, for 3D cone-beam circular scanning geometry, the linear system is "severely ill-posed" even if there are enough projections (Lionheart, 2013). Therefore, regularisation terms need to be introduced in real applications, as otherwise any noise (and there always is noise in real applications as mentioned above) will lead to very large errors in the reconstructed image. In view of the features of the system matrix, including sparsity, inconsistency, ill-conditioning, etc., an optimization criterion is set up with the system of equations. Iterative methods are then used to optimize the problem, which is a topic that goes back a long way (Forsythe, 1953).

After discretizing the CT projection process, the next step of iterative methods is the development of reconstruction algorithms. Such algorithms should be capable of handling the problem within the special mathematical environment of huge dimensions and sparseness. It should also be efficiently implementable on computers. The large size of $\mathbf{A}$ means that solving Eq.1.5 or Eq.1.6 directly is infeasible(Censor and Herman, 1987). Iterative algorithms solve them by simply using matrix-vector multiplications to update an image vector $\mathbf{x}_{rec}$, making it gradually approach the true solution. All iterative methods consist of three major steps which are repeated iteratively. First, a forward projection (FP) of the current estimated image $\mathbf{x}_{rec}$ creates artificial raw projection results which, in a second step, are compared to the real projection $\mathbf{y}$ in order to compute a residual. In the last step the correction term is back projected (BP) onto the volumetric object estimate (Oliveira et al., 2011; Tang et al., 2012; Zhang et al., 2006; Hsieh et al., 2013). The estimated image $\mathbf{x}_{rec}$ can start from anywhere. Generally, it starts from the zero space or a standard FBP reconstruction result (Beister et al., 2012).

The FP and BP mentioned above generally are the matrix-vector multiplications involving $\mathbf{A}$ and $\mathbf{A}^T$. Based on the portion of $\mathbf{A}$ used in each iteration, the iterative methods can be further divided into two categories, deterministic algorithms and random algorithms. The deterministic algorithms are algorithms which, given a particular initial value on image space, the output after a fixed number of iterations is determined. This is because in each iteration's FP and BP, the whole of $\mathbf{A}$ is used. This kind of CT reconstruction algorithm includes Landweber iteration, simultaneous iterative reconstruction technique (SIRT), component averaging (CAV), etc. (These algorithms will be introduced in the next Chapter). In the case of random algorithms, on the contrary, even with the same initial value on the image space, the output of the iteration is not necessarily the same after a determined number of iterations. This is because in each iteration's FP and BP, only a random part of $\mathbf{A}$ is used. This random selection causes the uncertainty

in the results, thus making the iteration unstable. Classical random algorithms include algebraic reconstruction technology (ART) with random sampling strategy, stochastic gradient descent (SGD), stochastic variance reduced gradient (SVRG), stochastic averaging gradient (SAG), etc. (These algorithms will be introduced in the next Chapter).

As mentioned above, when the linear model of CT scanning is severely ill-posed, for example, when the projection view's range is limited or the view is sparse, or when a 3D cone-beam circular scanning trajectory is adopted, a regularization term needs to be introduced to stabilise the solution. All iterative methods mentioned above can be used to optimize the objective function containing regularization terms such as Tikhonov regularization (Calvetti et al., 2000; Ying et al., 2004), total variation(TV) regularization (Borsic et al., 2001; Wang et al., 2017a), wavelet regularization (Verhaeghe et al., 2008; Belge et al., 2000) or early stopping regularization (Elfving et al., 2014). A detailed discussion will be introduced in the next chapter.

One drawback of iterative algorithms is their slow computation efficiency when performing FP and BP. This has become a bottleneck impeding the application of iterative methods in modern commercial CT scanners (Pan et al., 2009). With the development of X-ray detection technology, the resolution of both detectors and reconstructed images are getting increasingly higher. The increase in the amount of data makes the use of serial computing architectures, such as traditional CPUs, far too slow for tomographic reconstruction. Instead, graphic processing units (GPUs) are typically used as they have significant speed advantages over CPUs and thus are widely utilised to perform FP and BP computations (Kazantsev et al., 2013; Thompson and Lionheart, 2014). There are several mature toolboxes, such as TIGRE (Biguri et al., 2016) and ASTRA (van Aarle et al., 2015), that implement these operations on GPU as black-box operations and provide efficient interfaces to FP and BP operations for high level computing languages such as Matlab or Python.

1.3 The research objective of this project

Given that scientific and industrial tomographic imaging applications use increasingly larger and higher resolution detectors and require the use of more and more projections to scan an object, paired with the use of non-standard tomographic scanning trajectories, iterative methods are increasingly used in the reconstruction process. However, for large-scale tomographic reconstruction using computation nodes with limited storage capacity, the size of the projection data and the reconstructed image vector both exceed the maximum storage capacity of the computation nodes. As a result, both projection data and reconstructed image vector have to be partitioned into many smaller blocks. Since each iteration needs access to either all projection data or to the entire image (or to both), the computation nodes (e.g. GPUs) need to iterate over individual blocks

several times to compute individual FP and BP operations. This additional data access can significantly reduce reconstruction speed. As a result, the main research goal of this thesis is to improve iterative methods by reducing the data access to both projection data $\mathbf{y}$ and reconstructed image vector $\mathbf{x}$. Two algorithms that especially suit large-scale 3D image parallel reconstruction problems are proposed in this thesis, which are based on the latest developments in the field of random algorithms. The algorithms should be able to run efficiently on modern high-performance computing infrastructures, such as distributed computer networks equipped with GPUs, where each computation node does not have fast access to the entire dataset at once and where communication between different nodes is relatively slow.

There are three novel contributions in the thesis. Main contribution one is that the Coordinate-reduced Steepest Gradient Descent (CSGD) algorithm is proposed in the thesis, which is inspired by steepest gradient descent. CSGD is a distributed stochastic gradient descent algorithm specifically designed for very large tomographic inverse problems. In CSGD, the computation amount of projection can be easily tuned according to different parallel computation resources. When it is applied in parallel computation networks, the computation amount of each iteration is much less than the only available alternative approach, the block alternating direction method of multipliers (ADMM) (Parikh and Boyd, 2014), which has the same parallel computation architecture with CSGD and thus the reconstruction speed of CSGD are much faster than block ADMM.

Main contribution two is that based on the proposed CSGD and inspired by SAG (Schmidt et al., 2017), another parallel algorithm Block Stochastic Gradient Descent (BSGD) is proposed and it overcomes the original CSGD's drawback which does not approach the least square solution and the error variance of the calculated gradient does not converge to zero at the solution. Mathematical analysis shows that the fixed point of BSGD are at the least square solution, which is the best solution that minimises the error energy and is the maximum likelihood estimate. It has the same parallel computation architecture with the other gradient-based methods but the communication overhead is greatly reduced. Simulation shows that BSGD is of the fasted reconstruction speed when compared with CSGD and the other iterative methods which have been applied in the CT reconstruction field. Given the ill-conditioned nature of the tomographic reconstruction problem, simulations in the thesis show empirically, that BSGD can be embedded into a proximal optimisation algorithm and thus is able to compute regularised least squares solutions.

Main contribution three is the further exploration of the two proposed algorithms. The exploration mainly includes an importance sampling strategy and automatic parameter tuning trick. These two important tricks are very useful in terms of further accelerating two proposed algorithms. Besides, when the projection views are few and sparse, CSGD and BSGD are experimentally proved to be able to substitute gradient descent procedure in proximal gradient methods, which means that by combining CSGD/BSGD

with the TV denoising procedure, the objective function including a quadratic data fidelity term and a TV regularization term can be approximately optimized. Furthermore when solving the TV-based objective function, the TV denoising procedure can also be parallelized onto separate computation nodes by new proposed slicing methods. The last exploration is to apply BSGD and CSGD in realistic parallel networks (Iridis supercomputer in University of Southampton). The scalability of BSGD and CSGD are compared with each other and experiments verify that BSGD is faster than CSGD in realistic parallel computation environment. Apart from the synchronous application, the BSGD's asynchronous application property is also explored. The flexible communication increases each parallel node's independence and enables them to communicate with the master node without waiting for each other. Simulations have verified that under certain situations the asynchronous BSGD can achieve a faster reconstruction speed than synchronous BSGD.

This thesis consists of seven chapters. The second chapter is a literature review of the main iterative algorithms used in CT and a review of the latest parallel algorithms developed recently in optimization, which are the basis for our developments. Chapter 3 and 4 describe the two newly proposed algorithms CSGD and BSGD respectively. Chapter 5 explores the application of CSGD and BSGD in the optimization of TV norm regularised least squares problems. Chapter 6 illustrates the synchronous and asynchronous application of the two algorithms. Conclusions and suggestions for future work are included in Chapter 7. Appendix A presents some basic mathematical analysis of the two proposed algorithms.

1.4 Published papers

Some of work on this thesis has been published in conferences and peer reviewed journals. The following publications directly relate to the content of this thesis:

- "A parallel CT reconstruction algorithm using partial row and column blocks of the system matrix"(Gao and Blumensath, 2017). This is a short conference abstract based on the presentation at the conference TOSCA 2017, Portsmouth. This conference abstract firstly proposed the basic ideas of reconstructing the CT problem by using partial access to the projection data and the reconstruction volume, which is discussed in Chapter 3.
- "A Joint Row and Column Action Method for Cone-Beam Computed Tomography"(Gao and Blumensath, 2018b). In 2018 IEEE Transactions on Computational Imaging. This is a journal paper condensing the research in Chapter 3.
- "A parallel stochastic algorithm for large scale CT reconstruction"(Gao and Blumensath, 2019). This is a short conference abstract based on the presentation at

dXCT 2019, Huddersfield. The abstract proposed the basic algorithm in BSGD which is mainly described in Chapter 4.

- "BSGD-TV: A parallel algorithm solving total variation constrained image reconstruction problems"(Gao and Blumensath, 2018a). This is a short conference abstract published in 2018 iTWIST, Marseille. It presents the combination details of BSGD and TV-based denoising and shows that this combination can be used to approximately optimize the objective function containing TV regularizations. This part is discussed in Chapter 5.

Besides, there is another journal paper that is temporarily uploaded to arXiv(Gao et al., 2019).

- " Block stochastic gradient descent for large-scale tomographic reconstruction in a parallel network". This journal paper draft concludes the contents in Chapter 4.

Chapter 2

Literature review on iterative methods

As mentioned above, solving the linear equation directly is prohibitive due to the large size of the system matrix **A**. Iterative methods thus become mainstream tools to solve the inverse problem Eq.1.5. Unlike analytic methods which perform reconstruction within a single step using a specific inversion formula, iterative methods refine and modify the image iteratively in order to minimize an objective function. This objective function is based on the imaging system model and regulariser. Mathematically, the objective function tries to simultaneously optimize two different aspects of the reconstruction: the fidelity of reconstructed image data with measured projection data and the suppression of image noise by means of a regularization term that penalizes noisy solutions to the optimization problem (Stiller, 2018). Iterative reconstruction then consists of minimizing the objective function by repeated update of the image data, thus maximizing conformity between measured and reconstructed data, and minimizing image noise. The global iterative process is illustrated in Fig.2.1.

In this chapter, the iterative algorithms for reducing the date fidelity (i.e. the corresponding objective function does not contain regularization terms) are firstly introduced, followed by introductions of the optimization methods for objective functions containing regularizations terms.

2.1 Terminologies in algorithm descriptions

Before introducing the iterative algorithms, it is worthy to first explain the terms that are frequently appeared in the thesis.

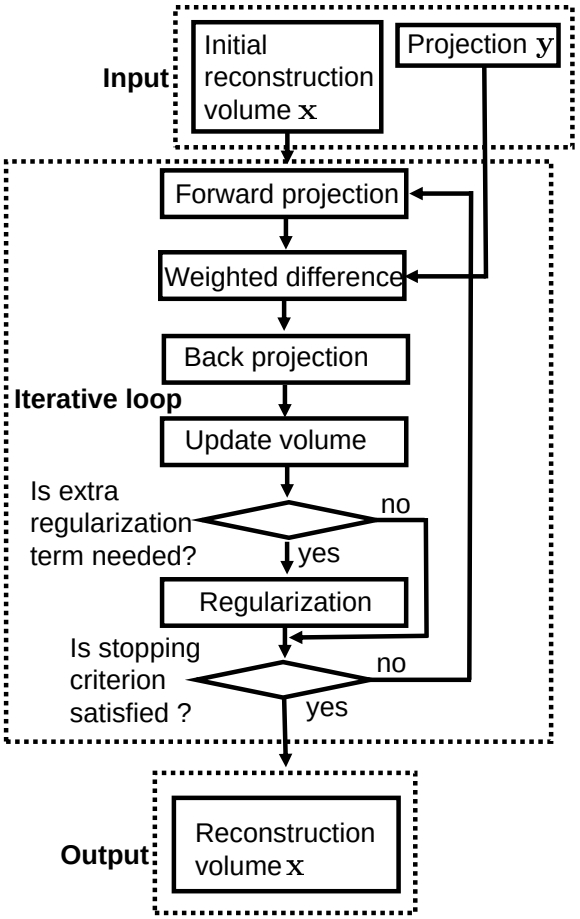


Figure 2.1: Basic scheme of iterative reconstruction process.

- iteration. An "iteration" will refer to the computational steps that are repeated in an iterative algorithm. Each iteration will compute updates of internal parameters and image.

- update direction. The update direction is the difference (possibly scaled) in the estimates of a quantity (such as the image we are trying to recover) before and after an iteration, that is, it is the vector that, when scaled and added to the previous estimate of a quantity, produces a new estimate. For example, in gradient descent method, the update direction is the minus gradient direction of current $\mathbf{x}_{rec}$. In the stochastic gradient descent methods, the update direction is stochastic but the expectation of this direction is an unbiased estimation of the gradient.

- step-length. It is the multiplier used to scale the update direction during an iteration. The μ is used as the step length in this thesis. It can be either a constant or a variable that changes in each iterations.

- epoch. A group of iterations during reconstruction. The number of iterations in each epoch depends on the algorithm and its parameters. For example, in this thesis, the gradient descent methods, which uses the whole of $\mathbf{y}, \mathbf{A}$ and update

the whole of $\mathbf{x}_{rec}$ in each iteration, are considered an epoch that only consists of one iteration. The stochastic gradient descent methods, using one row of $\mathbf{A}$ and $\mathbf{y}$ to calculate the update direction, are regarded as containing r iterations for one epoch. In the following k can be used to denote the iteration number but can also be used to denote epoch number. The important point is that in simulations, when a new $\mathbf{x}_{rec}^k$ is obtained, the reconstruction quality can be estimated and the simulations may be terminated when a stopping criterion has been satisfied.

- objective function. In this thesis, the most general form of objective function $F(\mathbf{x})$ is defined as $F(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$, where $f(\mathbf{x})$ is a data fidelity term and g(x) is a regulariser to stabilise the ill-posed problem. Generally the $f(\mathbf{x})$ is assumed as $\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2$. If the $g(\mathbf{x})$ adopts the TV regulariser, then $F(\mathbf{x})$ can be expressed as:

$$F(\mathbf{x}) = \underbrace{\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2}_{f(\mathbf{x})} + \underbrace{\lambda TV(\mathbf{x})}_{g(\mathbf{x})}, \tag{2.1}$$

  where the definition of $TV(\mathbf{x})$ is presented in section 2.3.3. As the optimisation of $F(\mathbf{x})$ is typically broken into separate steps, significant part of this thesis focus on the optimisation of $f(\mathbf{x})$.

- convergence. In this thesis, when talking about convergence, unless otherwise stated, we will mean the convergence in the euclidean norm of a sequence of estimates $\mathbf{x}^k$ towards the minimiser of the objective function. In terms of the convergence rate, it is the empirical rate of change of a cost function whilst it approaches a minimal value. Generally speaking, convergent iterations usually hint that a stable or approximately stable solution is gradually approached. This is usually reflected by the fact that the reconstructed image's quality no longer changes and finally become stable.

- reconstruction speed. Depending on the performance measure used, the reconstruction speed can be measured in terms of the change in the distance from the true image or in terms of the change of the cost function. In this thesis, it is an empirical measure of the computation speed to reach a certain level of error between the estimated image and the true image, which is reflected in the SNR trend. (defined in Eq.1.7)

- CPU. Central Processing Unit (CPU) in a computer is used to execute basic calculations. For example, in the thesis the update on $\mathbf{x}_{rec}$ is regarded as executed on CPUs.

- RAM. Random Access Memory (RAM) in a computer is the memory that exchange data with CPU. Its size is a limiting factor influencing the size of dataset to be addressed by a single computer.

- GPU. Graphics Processing Unit(GPU) is typically computationally faster than CPU. In the thesis, GPU can be easily used to execute intensive matrix-vector multiplications by using functions provided by existing CT reconstruction toolboxes.
- parallel network. It is a computer cluster containing several computation nodes. The computation nodes can be computers or GPUs. The proposed algorithms are specially designed for the parallel network with several computation nodes.
- master node. It is a node in a parallel network, which is responsible for data collection and distribution. Generally it is not responsible for intensive computation and is only used to the data storage ans some basic and simple computations.
- computation nodes. They are also named as servants or parallel nodes. They can execute code simultaneously with different dataset. Generally, the computation nodes calculate the intensive matrix-vector multiplications in this thesis.

2.2 Solving objective functions without regularization terms

Reducing data fidelity consists of the following three main steps: 1)Forward projection (FP) to produce synthesized projections of the current (partial of) image, which is often calculated as the matrix-vector using (partial of) $\mathbf{A}$ and (partial of) $\mathbf{x}$. 2) Estimation of the residual between the synthesized projections and the experimentally acquired projections $\mathbf{y}$. 3) Back projection (BP) of the weighted residual, which is often calculated as the matrix-vector using (partial of) $\mathbf{A}^T$ and (partial of) the weighted residual, to update the volume. The iterative cycle is repeated until a predefined stopping criterion is met, e.g. if a fixed number of iterations or a sufficiently small difference between the solutions of two subsequent iterative steps is reached.

2.2.1 Kaczmarz method

Kaczmarz method, also known as algebraic reconstruction technology (ART), was first proposed in 1937 (Sznajder, 2016) to solve Eq.1.5 and has been used as one of the most classical reconstruction methods in CT reconstruction since 1970 (Elfving et al., 2014). In fact, the first commercial CT scanner adopts this algorithm. To introduce Kaczmarz method, the linear system Eq.1.5 is expanded as

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + ... + a_{1c}x_c \\ &... \\ y_r &= a_{r1}x_1 + a_{r2}x_2 + ... + a_{rc}x_c. \end{aligned} \tag{2.2}$$

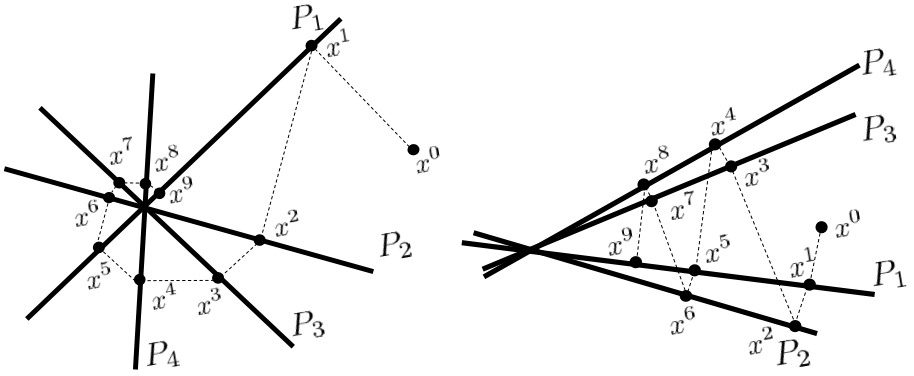
In Eq.2.2, each separate equation is called as a hyper-plane. For simplicity, the linear system is temporarily assumed to be consistent (i.e. $\mathbf{e} = \mathbf{0}$). The idea of the Kaczmarz type algorithms is to exploit the geometric structure of the Eq.2.2, and then using a sequential of projections to seek the solution. The recursive process at the k^{th} iteration can be formulated as follows

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{A}_i^{\mathrm{T}} \frac{\mathbf{A}_i \mathbf{x}^k - y_i}{\|\mathbf{A}_i\|^2}, \tag{2.3}$$

where y_i is the i^{th} single projection, $\mathbf{A}_i$ is the i^{th} row vector of the system matrix $\mathbf{A}$ and $\|\cdot\|$ is l_2 norm of a vector or matrix. In the most classical Kaczmarz, the i is sampled in a cyclic sequential way, i.e. $i = mod(k, r) + 1$, where $mod()$ is the function returning the remainder of a division. For a given $\mathbf{x}^k$, Eq.2.3 generates a $\mathbf{x}^{k+1}$ that satisfies the i^{th} equation in Eq.2.2. This update produces the following constrained optimization:

$$\arg\min_{\mathbf{A}_i \mathbf{x}^{k+1} = y_i} \|\mathbf{x} - \mathbf{x}^k\|_2^2. \tag{2.4}$$

Two geometric explanations of the above optimization when the linear system is consistent can be illustrated by Fig.2.2. It has illustrated that if the system is consistent



(a) consistent system (b) inconsistent system

Figure 2.2: Geometric illustrations of the classical Kaczmarz iterations with $m = 4$. $P_i, i = 1, 2, 3, 4$ is the hyper-plan formed by $\mathbf{A}_i \mathbf{x} = \mathbf{y}_i$.

(i.e. there is a common point for all hyper-planes), the Kaczmarz gradually approaches the solution. Furthermore, (Tanabe, 1971) has proved that regardless of the system is consistent or is not, the Kczmarz method converges to the solution $\mathbf{x}^\star = \mathbf{A}^\dagger \mathbf{y}$, where $\mathbf{A}^\dagger$ is the M-P inverse of matrix $\mathbf{A}$ (Sheng and Chen, 2010). According to the definition and property of $\mathbf{A}^\dagger$, if the system is consistent, then $\mathbf{x}^\star$ is the solution of the system and if the system is non-consistent, then $\mathbf{x}^\star$ can be the least square solution or the lest norm solution. By comparing the projection processes displayed in Fig.2.2, it is natural to have the intuition that convergence of the classical Kaczmarz algorithm highly depends on the geometric positions of the associated hyper-planes. If the normal vectors of every two successive hyper-planes keep reasonably large angles, the convergence of the classical

Kaczmarz algorithm will be fast, whereas two nearly parallel consecutive hyper-planes will make the convergence slow down. As a result, a good sampling sequence can accelerate the convergence speed of Kaczmarz methods. Another method to accelerate the convergence speed of Kaczmarz is to introduce relaxation parameter into the iteration. The iteration with relaxation is

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mu \mathbf{A}_i^{\mathrm{T}} \frac{\mathbf{A}_i \mathbf{x}^k - y_i}{\|\mathbf{A}_i\|^2}, \tag{2.5}$$

where the μ is the relaxation parameter. The use of relaxation parameters is important in practice. In the area of image reconstruction, it was demonstrated experimentally that small relaxation parameters significantly improve the practical performance of the Kaczmarz(Herman, 2009). However, the convergence property then becomes a bit complicated. When the system is consistent, it has been proved that when $0 < \mu < 2$, the relaxation variants of Kaczmarz still converge to the least norm solution of the system (Herman et al., 1978; Censor, 1981; Dai and Schön, 2015). When the system is not consistent, it has been proved that as the relaxation parameters go to zero, the relaxation variants of Kaczmarz approach a weighted least squares solution (Censor et al., 1983). This particular weighted least squares solution minimizes the sum of the squares of the Euclidean distances to the hyper-planes determined by the equations.

As indicated in Fig.2.2, convergence of classical Kaczmarz algorithm depends on the sequence of successive projections, which relies upon the ordering of rows in the matrix $\mathbf{A}$. In some real applications (Herman et al., 1978; Natterer, 2001), it is observed that instead of selecting rows of $\mathbf{A}$ sequentially at each step of the Kaczmarz algorithm, randomly selection can often improve its convergence. It has been proved that the rate of convergence can be significantly improved if the row index i in Eq.2.3 is sampled with probabilities $\frac{\|\mathbf{A}_i\|_2^2}{\|\mathbf{A}\|_F^2}$ (Strohmer and Vershynin, 2009). In terms of the convergence, if the system is consistent, the Kaczmarz with importance sampling converges to the $\mathbf{A}^\dagger\mathbf{y}$ in expectation (Bai and Wu, 2018). If the system is non-consistent, random Kaczmarz methods converges exponentially to the true original vector within a specified error bound, on expectation on the condition that $\mathbf{A}$ is of full column rank (Needell, 2010; Huang et al., 2020). Furthermore, it is found that the condition on the rank of $\mathbf{A}$ is not necessary, and proved that random Kaczmarz can converge to least norm solution on expectation, within a specified error bound (Zouzias and Freris, 2013). Random Kaczmarz can also be accelerated by relaxation parameter and the iteration is similar to Eq.2.5. The optimal relaxation parameter and the convergence rate of random Kaczmarz with relaxation under consistent and non-consistent situations is provided by (Moorman et al., 2020). It shows that when the system is consistent, the algorithm can still converge to the $\mathbf{A}^\dagger\mathbf{y}$ in expectation. Similarly, when the system is non-consistent, the sequence of iteration results converge to the $\mathbf{A}^\dagger\mathbf{y}$ within a specified error bound , which is controlled by the step length μ.

2.2.2 SIRT-type algorithms

The above discussed Kaczmarz method uses a "ray-by-ray" update strategy, which means that for every ray that is calculated, all pixel values associated with that ray are updated. SIRT-type methods distinguish themselves from Kaczmarz methods in that they do not update the iterated vector after each equation, but after an entire sweep through all the equations, and thus, during one sweep, they use the same residual vector for each equation. A rather general class of SIRT-type methods is given by

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mu^k \mathbf{C}\mathbf{A}^T\mathbf{R}(\mathbf{y} - \mathbf{A}\mathbf{x}^k), \tag{2.6}$$

where $\mathbf{C}$ and $\mathbf{R}$ are two positive definite diagonal matrices of orders c and r. This section mainly introduces four SIRT-type algorithms: SIRT, Cimmino's algorithm, CAV and Landweber iterative method. They distinguish themselves from each other by different weighting matrices $\mathbf{C}$ and $\mathbf{R}$.

1. SIRT

   SIRT is well known as it has been applied to a rather diverse set of reconstruction problems in medicine and biology since it was introduced more than four decades ago (Gilbert, 1972; Dilz et al., 2019). It updates each reconstructed image voxel by combining all single projection values whose corresponding X-rays passes through this voxel. It goes through all equations and then updates one image voxel at the end of each iteration using the average value of all computed changes for that imaging voxel. Taking the average correction value can suppress some interference factors, and the calculation result is independent of the pixel iteration order. SIRT updates are

   $$\forall j : x_j^{k+1} = x_j^k + \mu \frac{\sum_i \left[a_{i,j}\left(y_i - \sum_h a_{i,h}x_h^k\right) / \sum_h a_{i,h}\right]}{\sum_i a_{i,j}}, \tag{2.7}$$

   When it is rewritten into matrix-vector multiplication form shown in Eq.2.6, the $\mathbf{C}$ and $\mathbf{R}$ are then defined as:

   $$\begin{aligned} \tilde{C}_{j,j} &= \frac{1}{\sum_i a_{i,j}}, \\ \tilde{R}_{i,i} &= \frac{1}{\sum_j a_{i,j}}, \end{aligned} \tag{2.8}$$

   where $\tilde{C}_{j,j}$ and $\tilde{R}_{i,i}$ are the j^{th} and i^{th} diagonal element of $\mathbf{C}$ and $\mathbf{R}$ respectively.

   The main advantage of SIRT, especially compared with ART, is that the noise artefacts are more effectively suppressed and thus a soother reconstructed image with less artifacts is obtained (Guo and Devaney, 2005; Gregor and Fessler, 2015).

2. Cimmino's algorithm

Cimmino's algorithm can be viewed as a simultaneous application of Kaczmarz method. It projects the $\mathbf{x}^k$ to all hyper-planes and then average all single projection movements as the next general movement direction. The update can be rewritten as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \frac{\mu}{r} \sum_{i=1}^{r} \frac{y_i - \mathbf{A}_i \mathbf{x}^k}{\|\mathbf{A}_i\|^2} \mathbf{A}_i^{\mathrm{T}}, \tag{2.9}$$

When change Eq.2.9 into Eq.2.6, the $\mathbf{C}$ and $\mathbf{R}$ are changed as:

$$\begin{aligned} \mathbf{C} &= I, \\ \tilde{R}_{i,i} &= \frac{1}{r\|\mathbf{A}_i\|_2^2}. \end{aligned} \tag{2.10}$$

3. CAV algorithm

   One drawback of applying the Cimmino's algorithm in CT reconstruction is the slow convergence rate, which is caused by the sparsity of $\mathbf{A}$. Look back to Eq.2.9, it can be rewritten as

$$x_j^{k+1} = x_j^k + \frac{\mu^k}{r} \sum_{i=1}^{r} \frac{y_i - \mathbf{A}_i \mathbf{x}^k}{\|\mathbf{A}_i\|^2} a_{ij}. \tag{2.11}$$

   Since $\mathbf{A}$ is sparse, only a relatively "small" number of the elements $a_{1j}, a_{2j}, \cdots, a_{rj}$ are non-zero, but in Eq.2.11 the sum of their contributions is divided by the relatively "large" r, slowing down the progress of the algorithm. This observation leads to a modified algorithm "component averaging (CAV)' (Censor et al., 2001b)' which replaces r by the non zero elements for each column of $\mathbf{A}$. The iteration in CAV then becomes

$$x_j^{k+1} = x_j^k + \mu^k \sum_{i=1}^{r} \frac{y_i - \mathbf{A}_i \mathbf{x}^k}{\mathbf{A}_i^T \mathbf{S} \mathbf{A}_i} a_{i,j}, \tag{2.12}$$

   where $\mathbf{S} \in \mathbb{R}^{c \times c}$ is a diagonal matrix $diag(s_1, s_2, \cdots, s_c)$. s_j is the non zero element number for j^{th} column of $\mathbf{A}$. Eq.2.12 can also be expressed as a matrix-vector multiplication by setting:

$$\begin{aligned} \mathbf{C} &= I, \\ \tilde{R}_{i,i} &= \frac{1}{\mathbf{A}_i \mathbf{S} \mathbf{A}_i^T}. \end{aligned} \tag{2.13}$$

   Both the Cimmino's algorithm and CAV have been used in CT reconstructions and are shown to converge to a weighted least square solution. Compared with Cimmino's algorithm, the CAV method can significantly increase the reconstruction speed (Censor et al., 2001b), achieving the same image quality with Cimmino's algorithm by using less matrix-vector multiplications.

4. Landweber algorithm

   Traditional Landweber projection is to simply set $\mathbf{C}$ and $\mathbf{R}$ as unit matrices. It is also known as gradient descent(GD) method and its corresponding objective

function to be minimised is the quadratic objective function:

$$F(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{2.14}$$

GD updates $\mathbf{x}^k$ along the negative gradient direction, gradually reducing the gradient $\|\mathbf{g}\| = \|\mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{x}^k)\|$ to zero , leading $\mathbf{x}^k$ to the least-square solution $\mathbf{x}^\star = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$, which minimizes the residual norm(energy) $\|\mathbf{r}\|$.

For the objective function Eq.2.14, the maximum limit of step length μ is proportional to $\frac{1}{\|\mathbf{A}\|^2}$ (Gordon and Tibshirani, 2012). However, in large-scale CT scanning, calculating the l_2 norm of $\mathbf{A}$ is impossible due to the large computation overhead. As a result, the μ in realistic applications often needs to be repeatedly tuned until a value leading to a fast convergence rate is found. One method to avoid complicated parameter tuning is to choose μ so that the residual is reduced as much as possible in that direction. This is achieved by making $\mathbf{g}^k$ perpendicular to the next $\mathbf{g}^{k+1}$, as illustrated in Fig.2.3. Using $(\mathbf{g}^k)^T\mathbf{g}^{k+1} = 0$ and $\mathbf{x}^{k+1} = \mathbf{x}^k + \mu\mathbf{g}^k$,

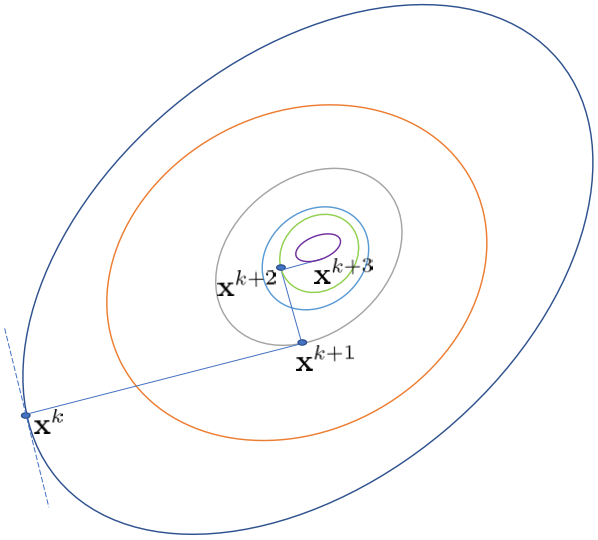


Figure 2.3: Different colours mean different equipotential of objective function Eq.2.14. The iterations approach the minimum of the objective function in a zig-zag manner, where the new search direction is orthogonal to the previous.

μ is:

$$\mu = \frac{\|\mathbf{g}^k\|^2}{\|\mathbf{A}\mathbf{g}^k\|^2}. \tag{2.15}$$

With the adaptive step size, it converges faster than GD with a constant μ, especially when μ is not well tuned. However, it requires an extra matrix-vector multiplication as well as a vector l_2 norm computation, so the computation overhead per iteration is increased compared to the GD with constant μ. The main drawback of GD is the slow convergence rate, which means that there often needs lots of iterations before the convergence. This is the nature of most SIRT-type algorithms (Van Scoy et al., 2017). A research direction thus becomes to accelerate

the convergence rate of GD. This section mainly introduces the most classical acceleration method, which is named as Nesterove's acceleration (Kim et al., 2013). Followed by a gradient descent iteration, $\mathbf{x}^{k+1}$ further "slides" along with a direction given by two adjacent iterations. The algorithm is shown in Algo.2.1.

Algorithm 2.1 Nesterov's accelerated GD in CT reconstruction area

1: Initialization: Determine the maximum allowed epoch number K_{max}. $\mathbf{x}^1 = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. μ is a constant step-length. $\tau^0 = 0$,$\lambda^0 = 0$
2: **for** $k = 1, 2, \cdots, K_{max}$ **do**
3: $\quad \lambda^k = \frac{1+\sqrt{1+4(\lambda^{k-1})^2}}{2}$
4: $\quad \tau^k = \frac{1-\lambda^{k-1}}{\lambda^k}$
5: $\quad \mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{x}^k$
6: $\quad \mathbf{g}^k = \mathbf{A}^T\mathbf{r}$
7: $\quad \tilde{\mathbf{x}}^{k+1} = \mathbf{x}^k + \mu\mathbf{g}^k$
8: $\quad \mathbf{x}^{k+1} = (1-\tau^k)\tilde{\mathbf{x}}^{k+1} + \tau^k\tilde{\mathbf{x}}^k$
9: **end for**
10: $\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

In general, the corresponding objective function for SIRT-type iteration can be viewed as a weighted quadratic objective function. Let $F_{\mathbf{R}}(\mathbf{x})$ be the following weighted least-squares objective function:

$$F_{\mathbf{R}}(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_{\mathbf{R}}^2 = \frac{1}{2}(\mathbf{y} - \mathbf{A}\mathbf{x})^T\mathbf{R}(\mathbf{y} - \mathbf{A}\mathbf{x}). \tag{2.16}$$

The gradient of $F_{\mathbf{R}}(\mathbf{x})$ is

$$\mathbf{g}_{\mathbf{R}}(\mathbf{x}) = -\mathbf{A}^T\mathbf{R}(\mathbf{y} - \mathbf{A}\mathbf{x}). \tag{2.17}$$

Then Eq.2.6 can be written as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mu^k\mathbf{C}\mathbf{g}_{\mathbf{R}}(\mathbf{x}). \tag{2.18}$$

Eq.2.18 implies that the minimisers for $F_{\mathbf{R}}(\mathbf{x})$, labelled as $\mathbf{x}^\star$, satisfy the equation $\mathbf{g}_{\mathbf{R}}(\mathbf{x}^\star) = \mathbf{0}$, i.e.,

$$\mathbf{A}^T\mathbf{R}\mathbf{A}\mathbf{x}^\star = \mathbf{A}^T\mathbf{R}\mathbf{y}, \tag{2.19}$$

which is called as normal equation. There is a solution of Eq.2.19 with the minimal $\mathbf{C}$ norm, i.e. a $\mathbf{x}^\star$ that has the minimal $\|\mathbf{x}^\star\|_{\mathbf{C}} = \mathbf{x}^{\star T}\mathbf{C}\mathbf{x}^\star$, which is denoted by $\mathbf{x}_{\mathbf{C},\mathbf{R}}(\mathbf{A}, \mathbf{y})$ (Ben-Israel and Greville, 2003). If we define $\|\mathbf{A}\|_{\mathbf{C},\mathbf{R}}$ as

$$\|\mathbf{A}\|_{\mathbf{C},\mathbf{R}} = sup_{\|\mathbf{x}\|_{\mathbf{C}}=1}\|\mathbf{A}\mathbf{x}\|_{\mathbf{R}}, \tag{2.20}$$

then it has been proved that when $0 \leq \mu \leq \frac{2}{\|\mathbf{A}\|_{\mathbf{C},\mathbf{R}}^2}$ and the initial iterated vector (i.e. $\mathbf{x}^0$) starts from $\mathbf{0}$, SIRT-type algorithms converge to $\mathbf{x}_{\mathbf{C},\mathbf{R}}(\mathbf{A}, \mathbf{y})$ (Jiang and Wang, 2003).

It is worthy to mention that there is a link between SIRT-type iteration and the least square problem. The general SIRT-type iteration in Eq.2.6 can be transformed and then it solves a quadratic objective function. To be more specific, let $\hat{\mathbf{y}} = \mathbf{R}^{\frac{1}{2}}\mathbf{y}$ and $\hat{\mathbf{x}} = \mathbf{C}^{-\frac{1}{2}}\mathbf{x}$ and $\hat{\mathbf{A}} = \mathbf{R}^{\frac{1}{2}}\mathbf{A}\mathbf{C}^{\frac{1}{2}}$ (NOTE: as $\mathbf{R}$ and $\mathbf{C}$ are diagonal and positive, the above matrix square-roots and inverses are well defined and easy to compute). The least squares problem $\|\hat{\mathbf{y}} - \hat{\mathbf{A}}\hat{\mathbf{x}}\|^2$ has a gradient descent algorithm that computes $\hat{\mathbf{x}}^{k+1} = \hat{\mathbf{x}}^k + \mu\hat{\mathbf{A}}^T(\hat{\mathbf{y}} - \hat{\mathbf{A}}\hat{\mathbf{x}})$ which, if we multiply both sides by $\mathbf{C}^{\frac{1}{2}}$, we obtain the Eq.2.6. Therefore, SIRT-type algorithms compute the least square solution to a linear problem with $\hat{\mathbf{y}} = \mathbf{R}^{\frac{1}{2}}\mathbf{y}$ and $\hat{\mathbf{x}} = \mathbf{C}^{\frac{1}{2}}\mathbf{x}$ and $\hat{\mathbf{A}} = \mathbf{R}^{\frac{1}{2}}\mathbf{A}\mathbf{C}^{\frac{1}{2}}$. As a result, in this thesis, the proposed algorithms are mainly designed to efficiently solve quadratic objective function Eq.2.14.

2.2.3 Block operations of iterative algorithms

The iteration in Kaczmarz is totally sequential since it uses one hyper-plane formed by a single X-ray to update the reconstructed vector. This property makes Kaczmarz method is of minor computation cost per update but is hard to be parallelised. On the contrary, the SIRT-type iteration uses all hyper-planes to update the whole of $\mathbf{x}$ by using the intact system matrix $\mathbf{A}$. According to the previous discussions, the FP and BP process can be parallelised in GPUs via mature reconstruction toolboxes. However, the computation cost per iteration can be enormous and unacceptable when the size of $\mathbf{A}$ is large, especially when the size of $\mathbf{x}$ and $\mathbf{y}$ both exceed the limited GPU memory. Block operations can be viewed as a moderate method between Kaczmarz and SIRT-type by dividing the $\mathbf{A}$ into many blocks and each iteration only uses one block of them. To be more specific, block operations on iteration algorithms can be further divided into row-action and column-action methods. Before introducing these methods, the partition methods on $\mathbf{A}$, as well as on $\mathbf{x}$ and $\mathbf{y}$, is firstly introduced.

2.2.3.1 Concepts of block partition

In modern CT scanning, the increasing size of $\mathbf{y}$ and $\mathbf{x}$ makes it difficult for the computation node, (e.g. GPU), to store all of the data within one data transmission. As a result, $\mathbf{y}$ and $\mathbf{x}$ are often divided and stored in many separate files and then sent to GPUs block by block. This property naturally benefits the application of block algorithms which only operate on parts of $\mathbf{y}$ and $\mathbf{x}$. This section mainly introduces the concepts and mathematical expressions of partitions.

Recall that the $\mathbf{A}$ is of r rows and c columns:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1c} \\ a_{21} & a_{22} & \dots & a_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ a_{r1} & a_{r2} & \dots & a_{rc} \end{bmatrix}. \tag{2.21}$$

Some basic definitions in the partition of system matrix $\mathbf{A}$ are concluded here: I_i is the i^{th} row index set, indexing m rows in $\mathbf{A}$. For example, $\mathbf{A}_{I_1}$ is the first row block of $\mathbf{A}$:

$$\mathbf{A}_{I_1} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1c} \\ a_{21} & a_{22} & \dots & a_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mc} \end{bmatrix}. \tag{2.22}$$

In this thesis, M is used to reflect the total row block number of $\mathbf{A}$. $\mathbf{A}$ with M row blocks can then be expressed as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{I_1} \\ \mathbf{A}_{I_2} \\ \vdots \\ \mathbf{A}_{I_M} \end{bmatrix}. \tag{2.23}$$

Similarly, J_j is the j^{th} column index set, indexing n columns in $\mathbf{A}$. For example, $\mathbf{A}^{J_1}$ means a column block of $\mathbf{A}$

$$\mathbf{A}^{J_1} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{r1} & a_{r2} & \dots & a_{rn} \end{bmatrix}. \tag{2.24}$$

N is used to reflect the total column block number of $\mathbf{A}$. $\mathbf{A}$ with N column blocks can be expressed as:

$$\mathbf{A} = \left[\mathbf{A}^{J_1}, \mathbf{A}^{J_2}, \cdots, \mathbf{A}^{J_N} \right]. \tag{2.25}$$

Besides, I_i and J_j are also the index set of corresponding X-ray measurements $\mathbf{y}$ and volume $\mathbf{x}$. Corresponding blocks are $\mathbf{y}_{I_i}$ and $\mathbf{x}_{J_j}$.

In the above example, the row blocks and column blocks all contain sequential row/column index numbers. In fact, arbitrary index sets are allowed as long as there is no overlap between row/column blocks.

If the row and column partitions are combined together, then the linear system Eq.1.5 is divided into MN blocks:

$$\begin{bmatrix} \mathbf{y}_{I_1} \\ ... \\ \mathbf{y}_{I_M} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{I_1}^{J_1}, \mathbf{A}_{I_1}^{J_2}, ..., \mathbf{A}_{I_1}^{J_N} \\ ... \\ \mathbf{A}_{I_M}^{J_1}, \mathbf{A}_{I_M}^{J_2}, ..., \mathbf{A}_{I_M}^{J_N} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{J_1} \\ ... \\ \mathbf{x}_{J_N} \end{bmatrix}, \tag{2.26}$$

where $\mathbf{A}_{I_i}^{J_j}$ is called a sub-matrix of $\mathbf{A}$. To illustrate the partition in both $\mathbf{y}$ and $\mathbf{x}$ better, a partition example in a 3D object is presented in Fig.2.4. In the figure, each separate projection view data is partitioned as a data block $\mathbf{y}_{I_i}$. In fact, the partition can be rather flexible. It can combine several projection views as a data block. Furthermore,the detector can be divided into several sub-detectors and take one sub-detector's corresponding projection data as a data block. This division method can be useful for the following proposed algorithms and will be illustrated in Chapter 3 and 4. In terms of the 3D volume partition, the volume can be split the along the central plane and then the total column block number is eight. It should be pointed out that the partition is generic and does not assume trajectory must be circular.

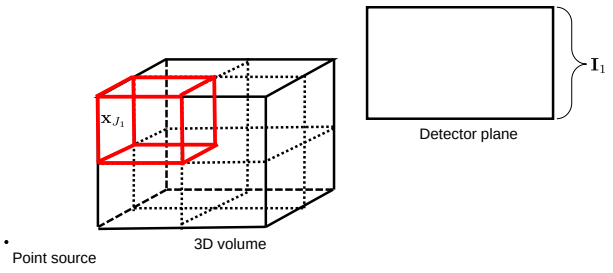


Figure 2.4: The 3D image is partitioned into eight sub-blocks, indexing from $\mathbf{x}_{J_1}$ to $\mathbf{x}_{J_8}$. The red frame sub-block is labelled as $\mathbf{x}_{J_1}$. One single projection view forms a separate row block and the index is $\mathbf{y}_{I_i}$. Here only illustrate the first row block index $\mathbf{y}_{I_1}$.

The partition on the projection data and the volume has important meaning when the dataset is of enormous size. For example, when using toolbox TIGRE (Biguri et al., 2016) or ASTRA (van Aarle et al., 2015) to calculate the FP result $\mathbf{y} = \mathbf{A}\mathbf{x}$, it is performed in a way that the toolboxes use arbitrary blocks and compute partial forward projections repeatedly, summing the results to compute the full projection. When using toolboxes to calculate BP, the process is similar to FP process. In this way the TIGRE or ASTRA can theoretically reconstruct arbitrary size of CT dataset with arbitrary scanning trajectories. To avoid having to circle through all blocks in each iteration, row-action and column-action methods are proposed and based on these two types algorithms we propose methods that update results after the computation with a single block.

In the following, I and J reflect a random element from set $\{I_i\}_{i=1}^{M}$ and set $\{J_j\}_{j=1}^{N}$ respectively. As a result, $\mathbf{y}_I$, $\mathbf{x}_J$ and $\mathbf{A}_I^J$ are random data blocks from sets $\{\mathbf{y}_{I_i}\}_{i=1}^{M}$,

$\{\mathbf{x}_{J_j}\}_{j=1}^{N}$ and $\{\mathbf{A}_{I_i}^{J_j}\}_{i=1,j=1}^{i=M,j=N}$. Within these notations, the introduction on row-action and column-action methods are followed.

2.2.3.2 Row-action methods

In some of the literatures, row-action methods are also called "ordered subset" methods (Xu et al., 2010; Guo and Chen, 2012). They divide the $\mathbf{A}$ and $\mathbf{y}$ into M row blocks and keep $\mathbf{x}$ undivided. Then the linear system in Eq.1.5 becomes:

$$\begin{bmatrix} \mathbf{y}_{I_1} \\ ... \\ \mathbf{y}_{I_M} \end{bmatrix} \simeq \begin{bmatrix} \mathbf{A}_{I_1} \\ ... \\ \mathbf{A}_{I_M} \end{bmatrix} \mathbf{x}. \tag{2.27}$$

According to (Elfving et al., 2017), the general row-action algorithm is shown in Algo.2.2. Here and in the following the random sample generally means that the row blocks can

Algorithm 2.2 Generic row-action iteration

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row indices into sets $\{I_i\}_{i\in[1,M]}$. $\mathbf{x}^1 \in \mathbb{R}^c$ is the arbitrary initial vector. $\mathbf{x}^k$ is the estimation of $\mathbf{x}$ in the k^{th} epoch. μ^i, $\mathbf{C}_i$ and $\mathbf{R}_i$ are the relaxation parameter and coefficient matrix for the i^{th} row block.
for $k = 1, 2, \cdots, K_{max}$ **do**
  $\tilde{\mathbf{x}}^1 = \mathbf{x}^k$
  for $i = 1, 2, ..., M$ (inner iterations) **do**
    Randomly or sequentially select I from $\{I_i\}_{i\in[1,M]}$
    $\tilde{\mathbf{x}}^{i+1} = \tilde{\mathbf{x}}^i + \mu^i \mathbf{C}_i \mathbf{A}_I^T \mathbf{R}_i (\mathbf{y}_I - \mathbf{A}_I \tilde{\mathbf{x}}^i)$
  end for
  $\mathbf{x}^{k+1} = \tilde{\mathbf{x}}^{M+1}$
end for
$\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

be either sampled with replacement or without replacement, and the possibility to be selected is the same among all participant blocks.

Row block applications can be applied on Kaczmarz algorithm. In the previous discussion, the Kaczmarz iteration only uses a single hyper-plane in each step, whilst the block Kaczmarz method selects multiple hyper-planes during one iteration (Needell and Tropp, 2014; Needell et al., 2015). The block Kaczmarz projects the current $\mathbf{x}$ onto the solution space $\mathbf{y}_I = \mathbf{A}_I \mathbf{x}$, where I is defined in section 2.2.3.1. The iteration then becomes (Needell and Tropp, 2014; Needell et al., 2015)

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{A}_I^\dagger (\mathbf{y}_I - \mathbf{A}_I \mathbf{x}^k), \tag{2.28}$$

where $\mathbf{A}_I^\dagger$ is a M-P inverse of $\mathbf{A}_I$. However, consider that in the large scale case, the calculation of $\mathbf{A}_I^\dagger$ can be infeasible and thus this kind of block application is not of

interest in this thesis. Compared with block Kaczmarz, the block version of SIRT and CAV are more feasible.

A block form of SIRT, which is also known as simultaneous algebraic reconstruction technique(SART) (Andersen and Kak, 1984), was propose in 1984. It separates the $\mathbf{A}$ and $\mathbf{y}$ based on the projection views and the definition of $\mathbf{C}_i$ and $\mathbf{R}_i$ in Algo.2.2 are similar to Eq.2.8 but is calculated based on $\mathbf{A}_I$ instead of $\mathbf{A}$. SART is regarded as a compromise between Kaczmarz and SIRT methods. Instead of treating each ray as a separate unit or treating them as a whole, the SART algorithm considers single projections within one projection view as a related system. Similarly, in the Block-iterative CAV (BICAV) (Censor et al., 2001a; Fernández et al., 2008), the weighting matrix $\mathbf{C}_i$ is still unit matrix whilst the $\mathbf{R}_i$ then counts the number of non-zero elements for each column of sub-matrix $\mathbf{A}_I$.

The block application of SIRT and CAV can significantly reduce the computation cost for each iteration, thus can meet the large data challenge to some extent. These algorithms have already been sealed as black boxes in the mainstream CT reconstruction toolboxes such as TIGRE and ASTRA.

The above introduction has included popular block algorithms when the weighting matrices $\mathbf{C}_i$ and $\mathbf{R}_i$ are not unit matrices. In the following of the thesis, the weighting matrices $\mathbf{C}_i$ and $\mathbf{R}_i$ will be simplified as unit matrices, and the objective function is the simple quadratic function shown in Eq.2.14. The main reason is that the weighted quadratic objective function can be transformed into a least square problem. The detailed explanations can be found in the end of section2.2.2.

Row-action type Landweber methods are also called as mini-batch stochastic gradient descent (mini-batch SGD). It can be seen that if the $M = r$ and the step length μ^i in Algo.2.2 is $\frac{1}{\|\mathbf{A}_i\|^2}$ then the mini-batch SGD becomes Kaczmarz method. The step length μ^i can also be a constant or a decreasing sequence, such type algorithms are called as SGD when $M = r$.

The idea to use a stochastic gradient instead of accurate gradient in optimization stems from the 1950s. It comes from the fact that the gradient $\mathbf{g}$ at $\mathbf{x}_{rec}$ of Eq.2.14 can be written as:

$$\mathbf{g} = \mathbf{A}^T(\mathbf{A}\mathbf{x}_{rec} - \mathbf{y}) = \sum_{i=1}^{r} \mathbf{g}_i = \sum_{i=1}^{r} \mathbf{A}_i^T(\mathbf{A}_i\mathbf{x}_{rec} - y_i). \tag{2.29}$$

It shows that the expectation of $\mathbf{g}_i$ is proportional to the true gradient

$$\mathbb{E}\mathbf{g}_i = \frac{1}{r}\sum_{i=1}^{r} \mathbf{g}_i = \frac{1}{r}\mathbf{g}. \tag{2.30}$$

Theoretical analysis established in various papers of SGD have guaranteed a convergent process when the random update direction is an unbiased estimation of the true gradient

(Bottou, 2010). A constant step length μ may help to accelerate the convergence rate at the initial stage but the final reconstruction results can be stuck at a relatively low precision level and it only converges to the least square solution within an error bound. In reality to ensure the convergence to the least square solution, the step length needs to decrease to 0 (Luo, 1991). The reason why a decreasing step length is required is because the stochastic direction (after being multiplied by r) in each iteration can be viewed as a true gradient direction being blurred by a stochastic noise vector. Although the expectation of the noise vector is zero, the variance of the noise vector does not go down when the iteration goes on. As a result, a mini-batch SGD is more popular than the original SGD in realistic applications by setting $1 < M < r$. The calculated stochastic gradient is the sum of several $\mathbf{g}_i$ and thus the variance of the noise vector is depressed (Ruder, 2016; Konečnỳ et al., 2015). Apart from using the mini-batch SGD, another method to reduce the variance is to accumulate the previous stochastic gradient, this method is called the incremental aggregated gradient (IAG) (Blatt et al., 2007) or stochastic averaging gradient (SAG) (Roux et al., 2012; Schmidt et al., 2017). These two algorithms are similar except that the previous method samples I from $\{I_i\}_{i\in[1,M]}$ cyclically while the later samples I randomly in each iteration. The SAG algorithm is shown in Algo2.3. It can be seen that SAG introduces new variables $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]}$.

Algorithm 2.3 SAG

Initialization: Determine the maximum allowed epoch number K_{max}. $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]}$ stores each stochastic gradient for different row blocks and initial values are zero matrices.
for $k = 1, 2, \cdots, K_{max}$ **do**
  Random select I_i from $\{I_i\}_{i\in[1,M]}$
  $\mathbf{r} = \mathbf{y}_{I_i} - \mathbf{A}_{I_i}\mathbf{x}^k$
  $\hat{\mathbf{g}}^i = \mathbf{A}_{I_i}^T\mathbf{r}$
  $\mathbf{g} = \sum_{i=1}^{M} \hat{\mathbf{g}}^i$
  $\mathbf{x}^{k+1} = \mathbf{x}^k + \mu\mathbf{g}$
end for
$\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

This means that storage demand of SAG is larger than for SGD, which is a drawback. However, both theoretical analysis and simulation results have verified that by recording each row block's stochastic gradient ingredients and summing them up, the accumulated $\mathbf{g}$ gradually approaches the true gradient direction. This property enables a constant step length μ ensuring the convergence to the least square solution and thus SAG has a faster convergence speed than SGD.

There is another famous algorithm to gradually reduce the error variance of estimated gradient, which is called Stochastic Variance Reduced Gradient (SVRG) (Johnson and Zhang, 2013). This algorithm not only converges with a constant step length but also does not require the storage of $\hat{\mathbf{g}}^i$. However, the cost is that it needs to calculate a full gradient with a predetermined frequency. The SVRG is shown in Algo.2.4. The idea is

Algorithm 2.4 SVRG

Initialization: Determine the maximum allowed epoch number K_{max}.
for $k = 1, 2, \cdots, K_{max}$ **do**
  $\tilde{\mathbf{x}} = \mathbf{x}^k$
  $\mathbf{g} = \nabla f(\tilde{\mathbf{x}}) = 2\mathbf{A}^T(\mathbf{A}\tilde{\mathbf{x}} - \mathbf{y})$
  $\mathbf{x}_{tem} = \tilde{\mathbf{x}}$
  for $t = 1, 2, \cdots, f$ **do**
    Randomly select I_i from $\{I_i\}_{i=1}^M$
    $\mathbf{x}_{tem} = \mathbf{x}_{tem} - \mu(M\nabla f_{I_i}(\mathbf{x}_{tem}) - M\nabla f_{I_i}(\tilde{\mathbf{x}}) + \mathbf{g})$,$\nabla f_{I_i}(\mathbf{x})$ is the stochastic gradient for $\frac{1}{2}\|\mathbf{y}_{I_i} - \mathbf{A}_{I_i}\mathbf{x}\|^2$.
  end for
  $\mathbf{x}^{k+1} = \mathbf{x}_{tem}$
end for
$\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

that although the variance of $\nabla f_{I_i}(\mathbf{x}_{tem})$ is high, the variance of difference $\nabla f_{I_i}(\mathbf{x}_{tem}) - \nabla f_{I_i}(\tilde{\mathbf{x}})$ decreases as the algorithm goes on. As a result, with more iterations, the update direction becomes close to the full gradient direction $\nabla f(\mathbf{x})$, hence eliminating its high error variance.

2.2.3.3 Column-action methods

In this section, the column-action methods, which are closely connected to coordinate descent (CD) optimization algorithms, are considered. The column-action methods divide the reconstructed image vector $\mathbf{x}_{rec}$ and $\mathbf{A}$ into N column blocks. The general column-action algorithm is shown in the Algo.2.5 (Elfving et al., 2017).

Algorithm 2.5 Generic column-action iteration

Initialization: Determine the maximum allowed epoch number K_{max}. Partition column indices into sets $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$. $\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{x}$. μ_j and $\mathbf{R}_j$ are the relaxation parameter and coefficient matrix for the j^{th} column block.
for $k = 1, 2, \cdots, K_{max}$ **do**
  $\tilde{\mathbf{x}}^1 = \mathbf{x}^k$
  for $j = 1, 2, ..., N$(inner iterations) **do**
    Select J_j as index J
    $\tilde{\mathbf{x}}_J^{j+1} = \tilde{\mathbf{x}}_J^j + \mu_j\mathbf{R}_j(\mathbf{A}^J)^T\mathbf{r}$
    $\mathbf{r} = \mathbf{r} - \mathbf{A}^J(\tilde{\mathbf{x}}_J^{j+1} - \tilde{\mathbf{x}}_J^j)$
  end for
  $\mathbf{x}^{k+1} = \tilde{\mathbf{x}}^{N+1}$
end for
$\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

The coordinate minimization algorithm is the most simple and intuitive one among coordinate descent families. It is especially popular when each iteration only updates

one pixel (i.e. $N = c$). The iteration in coordinate minimization is

$$\begin{aligned} \mu^k &= \frac{(\mathbf{A}^{J_j})^T \mathbf{r}}{\|\mathbf{A}^{J_j}\|^2}, \\ \mathbf{x}^{k+1} &= \mathbf{x}^k + \mu^k e_j, \\ \mathbf{r} &= \mathbf{r} - \mu^k \mathbf{A}^{J_j}, \end{aligned} \tag{2.31}$$

where e_j is a vector that the j^{th} element is 1 while the other elements are 0. Note that μ at the k^{th} iteration can be obtained by solving $e_j^T \nabla f(\mathbf{x}^k + \mu e_j) = 0$. It is easy to extend the preliminary single-update method into block form(Sauer et al., 1995; Fessler et al., 1997; Zheng et al., 2000; Benson et al., 2010; Fessler and Kim, 2011; Kim and Fessler, 2012). Block form CD can be further accelerated by randomly sampling blocks in each iteration, which is called Random Block Coordinate Descend (RBCD). The general RBCD iteration is

$$\begin{aligned} J_j &\in \{J_1, \cdots, J_N\} \\ \mathbf{x}_{J_j} &= \arg\min f(\mathbf{x}_{J_j}, \mathbf{x}_{\widehat{J}}), \end{aligned} \tag{2.32}$$

where $J_j \cup \widehat{J} = \{J_j\}_{j=1}^N$. The objective function Eq.2.14 can be expanded

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x} - \mathbf{y}^T \mathbf{A}\mathbf{x} + \frac{1}{2}\mathbf{y}^T \mathbf{y} \\ &= \frac{1}{2}(\sum_j \mathbf{A}^{J_j}\mathbf{x}_{J_j})^T(\sum_j \mathbf{A}^{J_j}\mathbf{x}_{J_j}) - \mathbf{y}^T(\sum_j \mathbf{A}^{J_j}\mathbf{x}_{J_j}) + \frac{1}{2}\mathbf{y}^T\mathbf{y}. \end{aligned} \tag{2.33}$$

According that Eq.2.32 only minimizes $\mathbf{x}_{J_j}$, it can be simplified as:

$$\begin{aligned} \arg\min f(\mathbf{x}_{J_j}, \mathbf{x}_{\widehat{J}}) &= \arg\min \frac{1}{2}(\mathbf{A}^{J_j}\mathbf{x}_{J_j})^T(\mathbf{A}^{J_j}\mathbf{x}_{J_j}) + \sum_{i\neq j}(\mathbf{A}^{J_i}\mathbf{x}_{J_i})^T(\mathbf{A}^{J_j}\mathbf{x}_{J_j}) - \mathbf{y}^T\mathbf{A}^{J_j}\mathbf{x}_{J_j} \\ &= \frac{1}{2}\arg\min(\mathbf{A}^{J_j}\mathbf{x}_{J_j})^T(\mathbf{A}^{J_j}\mathbf{x}_{J_j}) + \left[\sum_{i\neq j}(\mathbf{A}^{J_i}\mathbf{x}_{J_i})^T - \mathbf{y}^T\right]\mathbf{A}^{J_j}\mathbf{x}_{J_j}. \end{aligned} \tag{2.34}$$

The gradient direction of Eq.2.34 is

$$(\mathbf{A}^{J_j})^T\mathbf{A}^{J_j}\mathbf{x}_{J_j} + (\mathbf{A}^{J_j})^T\left[\sum_{i\neq j}(\mathbf{A}^{J_i}\mathbf{x}_{J_i})^T - \mathbf{y}^T\right] = (\mathbf{A}^{J_j})^T(\mathbf{A}\mathbf{x} - \mathbf{y}). \tag{2.35}$$

RBCD iteration then becomes:

$$\mathbf{x}_{J_j}^{k+1} = \mathbf{x}_{J_j}^k + \mu(\mathbf{A}^{J_j})^T(\mathbf{y} - \mathbf{A}\mathbf{x}), \tag{2.36}$$

where μ can be determined by line search and differs for different $\mathbf{x}_{J_j}$ (Qin et al., 2013) or it can simply be a constant (Lu and Xiao, 2015; Shalev Shwartz and Tewari, 2011).

Both pixel-based or block-based CD have been widely applied to CT reconstructions,

which can be traced back to the 1990s (Sauer and Bouman, 1993; Bouman and Sauer, 1996; Thibault et al., 2007). Many researches have focused on the selection sequence of image blocks. The selection criteria can be random (Hsieh et al., 2008; Chang et al., 2008; Richtárik and Takáč, 2014; Nesterov, 2012) or sequential(He and Buccafusca, 2016). Both forms show fast convergence of the high spatial frequencies and converge slowly at low spatial frequencies (Bouman and Sauer, 1996). When compared with other analytical algorithms as proposed in (De Man et al., 2005), CD methods have a rapid convergence rate when they are initialised with FBP result, which usually provide good estimations of the low spatial frequency of the image. Traditional CD methods just randomly select pixels or pixel groups (i.e. each block has the same probability to be selected), whilst an importance sampling strategy based on a calculated pixel selection criterion is proposed by (Yu et al., 2007, 2010). By non-homogeneously selecting pixel groups, the convergence rate of RBCD is increased by focusing on pixels which are in most need of an update. Another research interest is the partitioning of $\mathbf{x}$ space (Ha and Mueller, 2015; Benson et al., 2010; Fessler and Kim, 2011). However, most of them require more complicated computations, including solving a new symmetric linear system to determine which pixels should be updated simultaneously. Their iterations are no longer simple matrix-vector multiplications. As a result, this area is of less interest in this thesis.

2.2.3.4 Combinations of row and column-action methods

Algo.2.2 and Algo.2.5 have shown that full access to either $\mathbf{x}$ or $\mathbf{y}$ is required for row or column-action methods. For large-scale CT reconstruction where both the projection numbers and the pixel numbers of the reconstructed image are too large to be stored within one node, it is not fast enough to use either of these methods individually. Combinations of row-action and column-action methods was proposed to solve it. In general, these combined methods enable each iteration to only select a block of coordinates and to estimate the corresponding partial gradient based on a row block of system matrix $\mathbf{A}$ (Chen and Gu, 2016; Zhang and Gu, 2016). When applying these methods to the linear system, the general update scheme is

$$\mathbf{x}_J = \mathbf{x}_J - \mu \nabla_J f_I(\mathbf{x}), \tag{2.37}$$

where μ is the step-length and $f_I(\mathbf{x}) = \frac{1}{2}\|\mathbf{y}_I - \mathbf{A}_I\mathbf{x}\|_2^2$. $\nabla_J f_I(\mathbf{x})$ is $\mathbf{A}_I^J(\mathbf{y}_I - A_I\mathbf{x})$. Recently, several articles have discussed similar applications of this approach. In this thesis, these algorithms are uniformly called "stochastic block coordinate descent" (SBCD).

In particular, Wang (Wang and Banerjee, 2014) and Zhao (Zhao et al., 2014) independently proposed similar SBCD algorithms and both explored the application of variance reduction technique (Johnson and Zhang, 2013) to further accelerate the convergence. Konečnỳ (Konečnỳ et al., 2017) proposed a semi-stochastic coordinate descent method

to combine the stochastic gradient method and coordinate descent method to minimise a strongly convex problem. This method, however, requires the full gradient to be calculated in the first step. Xu (Xu and Yin, 2015) mathematically proved the convergence rate of SBCD when the step-length is decreasing and when the update strategy adopts Gauss-Seidel type, showing that the SBCD method has the same convergence rate as stochastic gradient methods when the objective function is convex. The sampling methods on row and column space is a research hotspot in SBCD type algorithms. For example, Shalev (Shalev Shwartz and Tewari, 2011) proposed a combination of SGD and RBCD methods to minimise l_1-regularized smooth convex problems by uniformly sampling the row blocks and non-uniformly selecting the column blocks. Leventhal (Leventhal and Lewis, 2010) combines the importance sampling scheme in Kaczmarz with RBCD method and established its iteration complexity. Chen (Chen and Gu, 2016) researched the application of SBCD in the sparsity constrained non-convex optimization by combining hard thresholding technique (Blumensath and Davies, 2009). Zhang and Lu separately proposed the optimal sampling method in the SBCD method, by randomly selecting column blocks and selecting row blocks based on a calculated probability (Zhang and Gu, 2016; Lu and Xiao, 2015) . Their methods can be regarded as a variant of the work in (Leventhal and Lewis, 2010).

Despite the fact that SBCD combines row-action and column-action methods. For each iteration, the computation of the gradient requires the accurate calculation of the block residue $\mathbf{r}_I = \mathbf{y}_I - \mathbf{A}_I\mathbf{x}$, which still requires access to all of $\mathbf{x}$. In other words, the system matrix $\mathbf{A}$ in SBCD type method is not actually separable in the column direction due to the need to calculate $\mathbf{r}_I$. This can be challenging for the computation nodes, for example, GPUs in a distributed network, whose capacity is limited.

2.2.4 Parallel applications of iterative algorithms

From a general point of view, Kaczmarz method belongs to a particular case of SGD with the learning rate equal to $\frac{1}{\mathbf{A}_i\mathbf{A}_i^T}$ (Kamath et al., 2015; Needell et al., 2014). As a result, the parallel algorithms designed for SGD are also suitable for the Kaczmarz method for CT reconstruction. For example, recently an Elastic Averaging SGD (EASGD) method was proposed (Zhang et al., 2015). It uses a parameter server architecture (Li et al., 2014), allowing each computation node to maintain its own local reconstructed $\mathbf{x}$. In the master node, there is an averaged vector $\widetilde{\mathbf{x}}$ which is linked with all computation nodes' parameters. The algorithm has both synchronous and asynchronous forms as well as a momentum form. Experiments show that the asynchronous form EASGD is stable and plausible under communication constraints. A similar parameter server architecture is also proposed in (Kamath et al., 2015), where a fusion centre assigns each computation node a block of system matrix $\mathbf{A}$ and the corresponding projection data block. Each node performs sequential Kaczmarz process and the fusion centre

performs a component average. This method requires synchronization between all nodes and the communication cost dramatically increases when the data size increases. To reduce the communication cost, a distributed randomized Kaczmarz was then proposed and makes the node only communicate with its neighbour, achieving an asynchronous communication between nodes. More discussions on asynchronous Kaczmarz or SGD methods can be found in (Liu et al., 2014; Recht et al., 2011; Zhao and Li, 2016).

The SIRT-type algorithm is widely applied in parallel CT reconstruction task (Li et al., 2005). Benson applied SIRT to a parallel framework to mitigate the sequential computational cost and observed the speed differences between pre-calculating the system matrix or calculating it on the fly when iteration is conducted on a distributed network (Benson and Gregor, 2005). Castro compared speed-up factors of SIRT and SART under different block numbers (Bilbao-Castro et al., 2004, 2006). Gregor improved the traditional SIRT method and the improved algorithm showing faster convergence rates than traditional SIRT in a parallel environment (Gregor and Benson, 2008). However, all of the literature mentioned above requires each node to keep a full copy of the reconstructed $\mathbf{x}$. Recently, Palenstijn proposed a method that requires each node only to keep a slice of the volume object (Palenstijn et al., 2015). This method requires a circular scanning trajectory and the block direction should be perpendicular to the scanning rotation axis. Each node is assigned with a sub-volume of the object and thus only a small area of the detector receives X-rays passing through this particular sub-volume. The iteration in Eq.2.6 of each node is independent except for the two neighbouring nodes which have overlapping projection areas. To obtain accurate forward projection data ($\mathbf{Ax}$ in the SIRT iteration), the communication thus only happens between such two nodes who share the overlapping area. In this parallel scheme, the communication cost between nodes decreases because of the reduced storage on $\mathbf{x}$ but heavily depends on the number of blocks. When the block number increases, the overlapping area also increases, which means that the communication load becomes heavier. As a result, the scalability of this parallel method is limited. Besides, this methodology is only suitable for standard circular scanning strategies, which limits its application . However, this method divides $\mathbf{x}$ into several blocks and assigns them to different nodes, which is an improvement to the previous method that requires each node to keep a full copy of $\mathbf{x}$. Concerning the parallel computation form BICAV, the process is similar to the block form SIRT, which adopts a parameter server form. A master node assigns different projection data into different computation nodes and each node performs BICAV. Finally, the master node communicates with all computation nodes through a weighted sum of all partial results (Bilbao-Castro et al., 2006).

Coordinate descent type algorithms, especially RBCD, are also suitable for parallel computation (Yu et al., 2006). A comprehensive convergence theory of parallel CD is established in (Richtárik and Takáč, 2016) and the theoretical speed-up is claimed as a simple expression depending on the number of parallel processors. In CT reconstruction,

the parallel CD is mainly studied in (Wang et al., 2016; Sabne et al., 2017; Wang et al., 2017b), focusing on the hardware application in CPU-GPU architecture and on the data transformation between GPU/CPU buffers. However, in mature reconstruction toolboxes adopted in this thesis, projection data transformation has already been sealed into the "black box" of matrix-vector multiplication operations. Research into detailed hardware data transformation is thus of less interest in this project.

2.3 Algorithms with regularizations

In tomography, the system matrix $\mathbf{A}$ in Eq.1.5 is always ill conditioned and the linear system is always ill-posed (Natterer, 2001). Since the solution of an ill-posed linear system is always severely influenced by the noise vector on the projection data, using Landweber projection or its block form without regularisation cannot obtain satisfying reconstruction results. This is because that the least square solution obtained by these methods, denoted by $\mathbf{x}^\star$ is not a meaningful approximation of the true solution(i.e. the image vector representing the scanned object). Regularization is thus often added when solving ill-posed linear system. In this section, three main regularization methods are introduced after explaining the semi-convergence property in the ill-posed system.

2.3.1 Semi-convergence and early stopping criteria

For the following discussion, the ill-posed linear system is rewritten as:

$$\mathbf{y} = \mathbf{A}\mathbf{x}_{true} + \mathbf{e} = \tilde{\mathbf{y}} + \mathbf{e}, \tag{2.38}$$

where the $\mathbf{x}_{true}$ is the scanned object, $\mathbf{e}$ is the error vector. According to Theorem 1.1 in (Elfving et al., 2010), SIRT-type iteration Eq.2.6 always converge to a solution of $\mathbf{x}^\star$ that minimises $\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_{\mathbf{R}}$ and thus $\mathbf{x}^\star$ is a weighted least square solution. When the noise vector is zero and the system is overdetermined, it is easy to prove that the $\mathbf{x}^\star \equiv \mathbf{x}_{true}$. However, if $\mathbf{e}$ is non-zero, then the story is different. Since the linear system is ill-posed and $\mathbf{A}$ is ill-conditioned, $\mathbf{x}^\star$ can be arbitrarily far away from $\mathbf{x}_{true}$. A brief explanation is presented here by using the Landweber projection as an example: According to the previous discussion, the Landweber projection, as well as its stochastic block version, converges (or converges on expectation) to the least square solution $\mathbf{x}^\star = (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{y}) = (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T(\tilde{\mathbf{y}} + \mathbf{e})) = \mathbf{x}_{true} + (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{e})$. For simplicity it is assumed that the original linear system is overdetermined and there is only one solution (i.e. the solution is the original scanned object vector $\mathbf{x}_{true}$) in the noise-free case. As $\mathbf{A}$ is ill conditioned, $\mathbf{A}^T\mathbf{A}$ is also ill conditioned and thus $(\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{e})$ is a large vector even if $\mathbf{e}$ is a small disturbance. As a result, during the iterations, although the distance of $\mathbf{x}^k$ to $\mathbf{x}^\star$ and the weighted residual norm (i.e. value of $F_{\mathbf{R}}(\mathbf{x})$

in Eq.2.16) decrease stably, the distance of $\mathbf{x}^k$ to $\mathbf{x}_{true}$, which reflects the reconstruction quality, will typically reduce initially but then increase. This phenomena is often called semi-convergence. In other words, when talking about semi-convergence, it does not mean the algorithm does not converge, instead, it still converge to $\mathbf{x}^\star$. However, in the discussion of semi-convergence, the reference solution is changed from $\mathbf{x}^\star$ to $\mathbf{x}_{true}$ and it is inevitable that the iteration will approach $\mathbf{x}_{true}$ and then move away from it. This phenomena widely exists in both Kaczmarz and SIRT-type iterations (Elfving et al., 2010, 2014). It is thus common to stop iterative algorithms after a few iterations, long before the method has converged to $\mathbf{x}^\star$. This strategy is called early stopping. Here the iteration number plays a regularization role.

(Elfving et al., 2010, 2014) have proposed the error analysis by dividing the difference between the realistic iterated vector and the ideal optimal solution (i.e. the result when dataset is noise free):

$$\mathbf{x}^k - \tilde{\mathbf{x}} = \mathbf{x}^k - \tilde{\mathbf{x}}^k + \tilde{\mathbf{x}}^k - \tilde{\mathbf{x}}, \tag{2.39}$$

where $\tilde{\mathbf{x}}$ is the minimiser of $\|\mathbf{A}\mathbf{x} - \tilde{\mathbf{y}}\|^2_{\mathbf{R}}$ (definition of $\tilde{\mathbf{y}}$ is in Eq.2.38) and $\tilde{\mathbf{x}} \equiv \mathbf{x}_{true}$ in overdetermined system. $\tilde{\mathbf{x}}^k$ is the iterated results (at k^{th} iteration) of Kaczmarz or SIRT-type algorithm when $\mathbf{e}$ is zero. Thus the difference is decomposed into two components: the noise error $\mathbf{x}^k - \tilde{\mathbf{x}}^k$ and the iteration error $\tilde{\mathbf{x}}^k - \tilde{\mathbf{x}}$. Kaczmarz and SIRT-type algorithms have been shown that their noise error is increasing and the error's upper bound is scaled by $\sqrt{k}$ and k respectively. On the contrary, the iteration error is always decreasing if the step length μ is set properly. (The discussion on proper μ is mentioned in section 2.2.) When the iterations are at the early stage, the noise error is negligible and thus the norm of the difference (i.e. $\|\mathbf{x}^k - \mathbf{x}_{true}\|$) decreases but when k increases to some point the noise error is large enough to make the $\|\mathbf{x}^k - \mathbf{x}_{true}\|$ increase. This mathematically explains the semi-convergence phenomena. It shows that this property widely exists as long as the system is ill-posed, i.e. a small change on $\mathbf{e}$ causes huge changes on $\mathbf{x}^\star$, making $\mathbf{x}^\star$ stay far away from $\mathbf{x}_{true}$.

Whilst SIRT and Kaczmarz converge to a limit point $\mathbf{x}^\star$, due to the ill conditioning of the CT inverse problem, the algorithms can provide solutions that are arbitrarily far away from $\mathbf{x}_{true}$. For ill-posed tomographic problems, regularisation is often achieved through filtering of the projections (as with the FBP or FDK algorithms) or through early stopping of iterative methods. Whilst these approaches can provide good results in practice, it is difficult to quantify the amount of regularisation and the quality of the solution. In this thesis, we thus take a different view on the solution of ill-posed tomographic reconstruction. We split the problem into two steps. 1) the explicit characterisation of a regularisation function $g(\mathbf{x})$ chosen suitably for our specific problem and 2) the efficient optimisation of the resulting cost function $f(\mathbf{x}) + g(\mathbf{x})$. There has been significant research into the selection of different regularisation terms and their suitability to tomographic problems. This thesis instead focuses entirely on the second problem,

the efficient solution of the optimisation problem when $f(\mathbf{x})$ is a least squares cost function. In fact, as discussed further below, optimisation of composite cost functions of the form $f(\mathbf{x}) + g(\mathbf{x})$ can often be done in two steps that deals with each term individually. We thus here deal primarily with the issue of efficient optimisation of $f(\mathbf{x})$ and then show how to use this approach in combination with a commonly used regulariser.

2.3.2 Tikhonov regularization

Ill-posed problems must be regularized if one wants to successfully achieve the task of numerically approximating their solutions. It is often said that the art of applying regularization methods is to maintain an adequate balance between a solution's accuracy and stability. There is a large body of work on regularization methods. Among all regularization methods, perhaps the best known and most commonly used method is the Tikhonov-Phillips method, which was originally proposed by Tikhonov and Phillips in 1962 and 1963 (Tikhonov, 1963). This method replaces the linear system Eq.1.5 by a penalized least-squares problem of the form (Donatelli et al., 2012):

$$F_(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2 + \lambda\|\mathbf{T}\mathbf{x}\|_2^2, \tag{2.40}$$

where $\lambda > 0$ is known as the regularization parameter, $\mathbf{T}$ is some suitably chosen Tikhonov matrix and a common choice is the unit matrix $\mathbf{I}$ or a matrix approximating the first or second order derivative operator (Hansen and O'Leary, 1993). Taking the $\mathbf{T} = \mathbf{I}$ as example. The minimiser of Eq.2.40 is

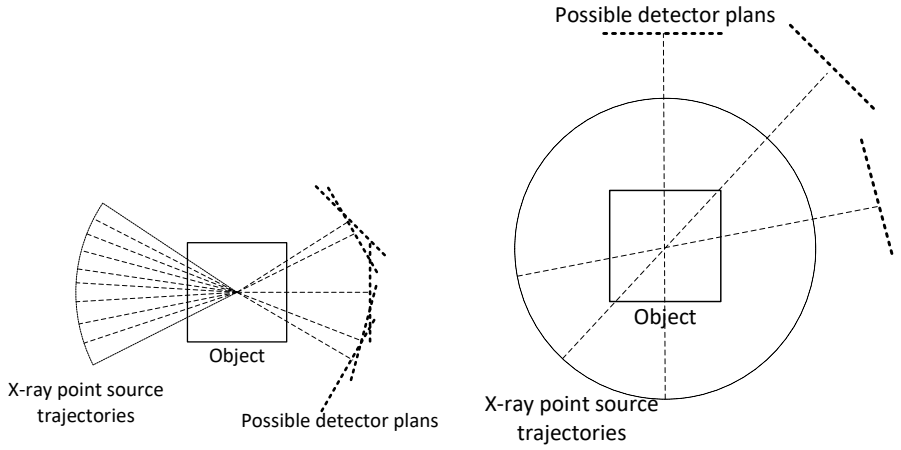
$$\mathbf{x}_\lambda^\star = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{y}, \tag{2.41}$$

and it can be iteratively approached by iterative shrinkage-thresholding algorithm (ISTA) or fast iterative shrinkage-thresholding algorithm (FISTA)(Beck and Teboulle, 2009b). It can be seen that the norm of the residual, i.e. $\|\mathbf{r}_\lambda\| = \|\mathbf{y} - \mathbf{A}\mathbf{x}_\lambda^\star\|$, is an increasing function of λ and the norm of $\mathbf{x}_\lambda^\star$ is the decreasing function of λ. The curve $L = \{\|\mathbf{x}_\lambda^\star\|, \|\mathbf{r}_\lambda\|\}$ is known as L-curve because under suitable conditions on $\mathbf{A}$ and $\mathbf{y}$ it is shaped roughly like the letter "L". The value λ that corresponds to the point $(\|\mathbf{x}_\lambda^\star\|, \|\mathbf{r}_\lambda\|)$ at the "vertex" of the "L" is denoted as λ_L. It is suggested to use λ_L as the optimal relaxation parameter in the reconstruction (Hansen, 1992). A heuristic motivation for this choice of λ is that when $\lambda > 0$ is "tiny", then the associated solution $\mathbf{x}_\lambda^\star$ has a large norm and is likely to be contaminated by the propagated error that stems from errors in the given projection data $\mathbf{y}$. Conversely, when λ is large, the vector $\mathbf{x}_\lambda^\star$ generally is a poor approximation of a solution of Eq.1.5 and the associated data fidelity $\|\mathbf{r}_\lambda\|$ is large. The choice $\lambda = \lambda_L$ seeks to best balance the data fidelity and the propagated error in the computed approximate solution $\mathbf{x}_\lambda^\star$.

2.3.3 Total variation regularizations

Generally, the Tikhonov regularization term prevents the pixels blowing up and it does not use much priori image information. This section introduces the total variation (TV) regularization, which uses the prior information such as the assumption that the gradient of the image is sparse.

In many real-world applications, projection data can only be obtained at imperfect angle ranges due to constraints of data acquisition time or constraints of the scanning geometry, which both cause incomplete data problems. An incomplete dataset is mainly caused by two reasons: one is limited-views scanning and the other one is the few-view scanning (or sparse-views scanning). A limited-views problem means that the scan angle range is small (e.g., 90° coverage or even less). For example, in industrial non-invasive detection, when the object size is large, the projection angle range can be rather limited because it is infeasible to rotate around the large volume (Banjak et al., 2016). The few-views problem refers to the situation that a full scanning view range (360° for cone beam scanning and 180° for parallel scanning) is achieved, but there is a large gap between adjacent projection views. For example, in medical scanning, the interval of the projection angle is supposed to be large, aiming to reduce the side effect caused by X-ray dose (Tian et al., 2011). An illustration of limited-views problem and few-views problem is shown in Fig.2.5. Image reconstruction using an incomplete



(a) Limited-views problem (b) Few-views problem

Figure 2.5: Two main incomplete datasets in 2D CT scanning.

dataset has been a hot research topic in recent years. Iterative algorithms based on the minimization of total variation (TV) constraints is of particular interest. The TV constraint can be derived from the compressed sensing (CS) theory proposed in (Cand and Wakin, 2008), which can achieve an exact recovery of an image from sparse samples of its discrete Fourier transform. The exact recovery depends on the fact that there exists some representation of the image for which the corresponding coefficients are

sparse. Although in many realistic applications the image function of the object to be detected is not sparse, it often has the characteristic of an approximate slice continuity. As a result, the gradient magnitude of the image is sparse. For images with sparse gradient properties and insufficient projections, according to the CS theory, images can be reconstructed by minimizing the total variation of the image (i.e., the l_1 norm of the gradient image) while subjecting to a data fidelity condition. For a discrete 2D image $\mathbf{x} \in \mathbb{R}^{K\times K}$, the basic TV regularizer is

$$\begin{aligned} TV(\mathbf{x}) &= \|\Delta f\|_1, \\ &= \sum_{2\leq i,j\leq K} \sqrt{(x_{i,j} - x_{i-1,j})^2 + (x_{i,j} - x_{i,j-1})^2}, \end{aligned} \tag{2.42}$$

where $TV(\mathbf{x})$ is a function calculating the total variation of $\mathbf{x}$ when it is expanded into 2D or 3D image forms and $x_{i,j}$ is the pixel intensity at i^{th} row and j^{th} column of 2D image $\mathbf{x}$. For the 3D case, the TV norm is similar to Eq.2.42 and an extra dimension is added. In limited-view problems, simulations and theoretical analysis both demonstrate that artifacts in the reconstructed images are directional and the basic TV norm in Eq.2.42 is unable to effectively eliminate the directional artifacts (Chen et al., 2013; Jin et al., 2010). Consequently, the TV regularizer is re-designed. Detailed discussions can be found in(Chen et al., 2013; Jin et al., 2010; Islam, 2013; Kongskov and Dong, 2017; Wang et al., 2017a).

Total variation regularizer was first proposed for image denoising in (Rudin et al., 1992) and then extended to image deblurring in (Rudin and Osher, 1994). In comparison to the well known Tikhonov regularizers, TV regularizers can better preserve sharp edges or object boundaries that are usually the most important features to recover. TV-based algebraic reconstruction then optimises the cost function

$$\begin{aligned} &\mathbf{x} = \arg\min_{\mathbf{x}_{rec}} TV(\mathbf{x}_{rec}), \\ &\text{subject to} \|\mathbf{y} - \mathbf{A}\mathbf{x}_{rec}\|_2^2 \leq \epsilon, \end{aligned} \tag{2.43}$$

or equivalently (Sidky and Pan, 2008)

$$\mathbf{x} = \arg\min_{\mathbf{x}_{rec}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}_{rec}\|_2^2 + \lambda TV(\mathbf{x}_{rec}), \tag{2.44}$$

where λ is a relaxation parameter and ϵ reflects the data inconsistency caused by noise vector $\mathbf{e}$. The optimization problems cannot easily be solved using standard gradient due to the non-smoothness of the objective function. Many algorithms have been proposed in the literatures. In this section these algorithms are categorized into derivative method and proximal method.

2.3.3.1 Derivative method

In the earliest TV constrained CT reconstruction, Sidky proposed an ART-TV algorithm based on the combination of the steepest gradient descent of the TV norm and the Kaczmarz method (Sidky et al., 2006). The principle of the algorithm is as follows: during the projection process, the image is roughly obtained by the Kaczmarz method shown in Eq.2.3. After a complete cycle of update steps, a steepest descent iteration reducing the TV norm is carried out. The basic algorithm is shown in Algo.2.6.

Algorithm 2.6 ART-TV algorithm

Initialization: Determine the maximum allowed epoch number K_{max}. $\mathbf{x}^0 \in \mathbb{R}^c$ is the arbitrary initial vector. The ART($\mathbf{x}$) means a whole Kaczmarz iterations through all system matrix rows. λ is a predefined parameter to control the TV decreasing step-length. t_{max} is the maximum loop number for TV decreasing process.

for $k = 1, 2, \cdots, K_{max}$ **do**
  $\mathbf{x}_{old} = \mathbf{x}^k$
  $\mathbf{x}_{new} = ART(\mathbf{x}^k)$
  $d_A = \|\mathbf{x}_{new} - \mathbf{x}_{old}\|_2$
  for $t = 1, 2, ..., t_{max}$ **do**
    $\mathbf{V}_{(i,j)} = \frac{\partial TV(\mathbf{x}_{new})}{\partial x_{new(i,j)}}$
    $\mathbf{x}_{new} = \mathbf{x}_{new} - \lambda d_A \mathbf{V}$
  end for
  $\mathbf{x}^{k+1} = \mathbf{x}_{new}$
end for
$\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

In terms of the calculation of $\mathbf{V}$ in Algo.2.6, since $TV(\mathbf{x})$ is not differentiable everywhere, the derivative method uses an approximated smoothed formula and obtains the partial derivative function:

$$\begin{aligned}\frac{\partial TV(\mathbf{x})}{\partial x_{(i,j)}} \approx & \frac{2(x_{i,j} - x_{i-1,j}) + 2(x_{i,j} - x_{i,j-1})}{\sqrt{\xi + (x_{i,j} - x_{i-1,j})^2 + (x_{i,j} - x_{i,j-1})^2}} - \\ & \frac{2(x_{i+1,j} - x_{i,j})}{\sqrt{\xi + (x_{i+1,j} - x_{i,j})^2 + (x_{i+1,j} - x_{i+1,j-1})^2}} - \\ & \frac{2(x_{i,j+1} - x_{i,j})}{\sqrt{\xi + (x_{i,j+1} - x_{i,j})^2 + (x_{i,j+1} - x_{i-1,j+1})^2}},\end{aligned} \tag{2.45}$$

where ξ is a non zero minimal value to make the denominator non-zero.

The ART-TV is easy to implement and many variants have been proposed during the last two decades (Sidky and Pan, 2008; Herman and Davidi, 2008; Yu and Wang, 2009; Sidky et al., 2011; Chen et al., 2013; Islam, 2013). One improvement algorithm, adaptive steepest descent projection onto convex sets (ASD-POCS) (Sidky and Pan, 2008), overcomes the assumption of ART-TV which requires the linear system to be consistent (i.e. $\mathbf{e} = \mathbf{0}$), and considers the situation when the linear system is inconsistent by adopting

an adaptive step-length to control the reduction of the TV norm. The ART-TV method can also be improved by using Block-ART (Herman and Davidi, 2008) or Block-SART (Yu and Wang, 2009) methods to substitute the original ART.

2.3.3.2 Proximal method

It is worthy to stress that the proximal methods are an important motivation for the thesis, as they allow to solve regularised least squares estimation problems by alternating between least squares estimation (which can be done efficiently with newly proposed algorithms) and a proximal operation to enforce the constraint. Before introducing proximal methods, some basic concepts of proximal operator and proximal method's corresponding mathematical model are first introduced.

A function is called convex if the line segment between any two points on the graph of the function lies above the graph between the two points. Classical TV regularization term is proved to be a convex function (Selesnick et al., 2020). One important property of convex function is that a strictly convex function on an open set has no more than one minimum.

The proximal method can quickly solve a convex optimization problem: the objective function $F(\mathbf{x})$ is not differentiable everywhere but can be divided as a sum of convex-differentiable function $f(\mathbf{x})$ and a convex but not necessary differentiable function $g(\mathbf{x})$. i.e.

$$\mathbf{x}^{\star} = \arg\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = \arg\min_{\mathbf{x}} [f(\mathbf{x}) + g(\mathbf{x})], \tag{2.46}$$

Traditional GD-type is only able to achieve the minimization of differentiable function, and thus GD-type algorithms are not suitable for the optimization of Eq.2.46 due to the possibly non-differentiable function $g(\mathbf{x})$. For this kind of mathematical model, a proximal operator is introduced

$$prox_{\mu}(g)(\mathbf{x}) = \arg\min g(\mathbf{u}) + \frac{1}{2\mu}\|\mathbf{u} - \mathbf{x}\|_2^2. \tag{2.47}$$

The proximal operator is only designed for the non-differentiable function $g(\mathbf{x})$ and is irrelevant to the differentiable function $f(\mathbf{x})$. Eq.2.47 can be explained as given a vector $\mathbf{x}$, find another vector $\mathbf{u} = prox_{\mu}(g)(\mathbf{x})$. By making $g(\mathbf{u}) + \frac{1}{2\mu}\|\mathbf{u} - \mathbf{x}\|_2^2$ minimal, the $\mathbf{u}$ makes the value of $g(\mathbf{u})$ small and is located near the original vector $\mathbf{x}$.

To deepen the understanding of proximal operator, here discuss three special and simple situations for function $g(\mathbf{x})$: 1) $g(\mathbf{x}) = 0$, 2)$g(\mathbf{x}) = \|\mathbf{x}\|_1$, 3) $g(\mathbf{x}) = \|\mathbf{x}\|_2^2$ to further illustrate the proximal operator. When $g(\mathbf{x}) = 0$, the proximal operator $prox_{\mu}(g)(\mathbf{x}) = \arg\min 0 + \frac{1}{2\mu}\|\mathbf{u}-\mathbf{x}\|_2^2 = \mathbf{x}$. When the $g(\mathbf{x}) = \|\mathbf{x}\|_1$, the proximal operator $prox_{\mu}(g)(\mathbf{x}) =$

$\arg\min \|\mathbf{u}\|_1 + \frac{1}{2\mu}\|\mathbf{u}-\mathbf{x}\|_2^2 = S_\mu(\mathbf{x})$. Using $[S_\mu(\mathbf{x})]_i$ to express the i^{th} element of $S_\mu(\mathbf{x})$:

$$[S_\mu(\mathbf{x})]_i = \begin{cases} \mathbf{x}_i - \mu, & \mathbf{x}_i \geqslant \mu \\ 0, & -\mu < \mathbf{x}_i < \mu \\ \mathbf{x}_i + \mu, & \mathbf{x}_i \leqslant -\mu, \end{cases} \tag{2.48}$$

which is called as soft thresholding function. When $g(\mathbf{x}) = \|\mathbf{x}\|_2^2$, since it is differentiable, it is easy to show that $prox_\mu(g)(\mathbf{x}) = \arg\min \|\mathbf{u}\|_2^2 + \frac{1}{2\mu}\|\mathbf{u}-\mathbf{x}\|_2^2 = \frac{1}{1+2\mu}\mathbf{x}$.

The proximal method to iteratively optimize Eq.2.46 is expressed as (Beck and Teboulle, 2009b; Kamilov, 2016; Rose et al., 2014; Combettes and Pesquet, 2011):

$$\mathbf{x}^{k+1} = prox_\mu(g)(\mathbf{x}^k - \mu\nabla f(\mathbf{x}^k)), \tag{2.49}$$

where the μ is not only step length to reduce the data fidelity of $f(\mathbf{x})$ but is also the footage parameter in the proximal operator. For this iteration, it can be explained that given the current k^{th} iteration result $\mathbf{x}^k$, first reduce $f(\mathbf{x})$ by moving $\mathbf{x}^k$ along with the minus gradient direction with a step length μ and obtain an intermediate updated value $\tilde{\mathbf{x}}^k$. Based on $\tilde{\mathbf{x}}^k$, an $\mathbf{u}$ is found by using proximal operator. The found $\mathbf{u}$ makes the non-differentiable function g small enough and is close to the intermediate updated value $\tilde{\mathbf{x}}^k$. The $\mathbf{u}$ is thus used as the next iteration result $\mathbf{x}^{k+1}$.

Eq.2.44 can be changed into Eq.2.46 if $f = \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}_{rec}\|_2^2$ and $g = \lambda TV(\mathbf{x}_{rec})$. This is because that the quadratic objective function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|^2$ and the TV definition $TV(\mathbf{x})$ are both convex functions. However the TV norm is not differentiable everywhere. This can be easily explained by the Eq.2.45. It can be seen that if $\xi = 0$ and $x_{i,j+1} = x_{i,j}$ then the denominator is zero.

The general iteration for TV-based optimization problem can be regarded as alternatively performing a gradient descent step and then applying the proximal operator on the iterated intermediate result to reduce the TV norm. To be more specific, an intermediate updated value $\tilde{\mathbf{x}}^k$ is first obtained after a GD step, and then the proximal operator is applied on $\tilde{\mathbf{x}}^k$ as:

$$\begin{aligned} prox_\mu(g)(\mathbf{x}) = prox_\mu(\lambda TV)(\mathbf{x}) = \arg\min \lambda TV(\mathbf{u}) + \frac{1}{2\mu}\|\mathbf{u}-\mathbf{x}\|_2^2 \\ \arg\min 2\mu\lambda TV(\mathbf{u}) + \|\mathbf{u}-\mathbf{x}\|_2^2 \end{aligned} \tag{2.50}$$

Unlike previous mentioned $g(\mathbf{x}) = 0$, $\|\mathbf{x}\|_1$ or $\|\mathbf{x}\|^2$ case, when $g(\mathbf{x}) = \lambda TV(\mathbf{x})$, there is no simple and direct solution of Eq.2.50. According to the discussion in Eq.4.1 in (Beck and Teboulle, 2009a) , the Eq.2.50 is a TV-based denoising problem and need to be solved by an iterative algorithm. As a result, the proximal method to optimize the TV-based regularization model Eq.2.44 is divided into two steps: The first step is to reduce the data fidelity of quadratic objective function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|^2$ with step length

μ and obtain an intermediate image vector $\tilde{\mathbf{x}}^k$. The second step is to perform a TV-based denoising algorithm to denoise the intermediate vector $\tilde{\mathbf{x}}^k$ with the regularization parameter $\mu\lambda$. The pseudocode of the TV-based denoising algorithm can be referenced in (Beck and Teboulle, 2009a).

To conclude, the early stopping criteria, Tikhonov regularization and TV regularization are all widely applied in the iterative CT reconstructions. Early stopping criteria stops the iteration before they converge, ensuring that the projection error does not propagate much. The Tikhonov and TV regularizations introduce some prior knowledge of the image. The Tikhonov regularization prevents the pixel value of reconstructed image blow up and stabilises the solution when the projection data is noisy and the linear system is severely ill-posed. The TV regularization term is not differentiable, which is different from the Tikhonov and thus the traditional gradient descent method is not suitable for TV-based optimization. Instead, a proximal operator is introduced. The iteration then includes a gradient descent step to reduce the data fidelity and then followed by a TV-based denoising procedure. Compared with Tikhonov regularization, TV regularization introduces more prior information by considering the slice continuity of the image and it is widely used in the case when the projection views are sparse or limited.

2.4 Other related method

Solving linear system Eq.1.5 is not only constrained to row-action and column-action methods. There are other algorithms that are available for general linear inverse problem. In this section, Block Alternating Direction Method of Multipliers" (block ADMM), which is widely used in machine learning, is briefly introduced. It shows that it can be applied in the large scale CT reconstruction cases.

ADMM is a method to solve the decomposable convex optimization problem. It is very effective in solving large-scale problems. Using the ADMM algorithm, the original objective function can be decomposed equivalently into a number of sub-problems. ADMM then solves each sub-problem in parallel, and then combines the solution of the sub-problems to find a global solution of the original problem. ADMM was originally proposed in 1975 and was re-examined by Boyd, who proved that the ADMM is suitable for large-scale distributed optimization problems (Boyd et al., 2011). After that the ADMM has been widely applied in CT reconstruction (Chen et al., 2014; Chun et al., 2014; Wang et al., 2019). The block version of ADMM (Parikh and Boyd, 2014) enables the total separation on the column direction of $\mathbf{A}$ and the partition of the whole linear

system as

$$\begin{bmatrix} \mathbf{y}_{I_1} \\ \mathbf{y}_{I_2} \\ \vdots \\ \mathbf{y}_{I_M} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{I_1}^{J_1} & \mathbf{A}_{I_1}^{J_2} & \cdots & \mathbf{A}_{I_1}^{J_N} \\ \mathbf{A}_{I_2}^{J_1} & \mathbf{A}_{I_2}^{J_2} & \cdots & \mathbf{A}_{I_2}^{J_N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{I_M}^{J_1} & \mathbf{A}_{I_M}^{J_2} & \cdots & \mathbf{A}_{I_M}^{J_N} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{J_1} \\ \mathbf{x}_{J_2} \\ \vdots \\ \mathbf{x}_{J_N} \end{bmatrix}. \tag{2.51}$$

Each computation node, in the distributed network, only receives $\mathbf{A}_{I_i}^{J_j}$, $\mathbf{x}_{J_j}$ and $\mathbf{y}_{I_i}$ and no longer needs the calculation of $\mathbf{r}_I$.

Before introducing the block version of ADMM, it is necessary to give a brief instruction on the ADMM iteration scheme. The ADMM method can solve a general convex optimisation problem

$$\begin{aligned} &\min f(\mathbf{z}) + g(\mathbf{x}), \\ &\text{subject to} \quad \mathbf{z} = \mathbf{A}\mathbf{x}, \end{aligned} \tag{2.52}$$

with variables $\mathbf{x} \in \mathbb{R}^c$ and $\mathbf{z} \in \mathbb{R}^r$, where $f : \mathbb{R}^r \longrightarrow \mathbb{R}$ and $g : \mathbb{R}^c \longrightarrow \mathbb{R}$. The basic iteration of ADMM follows:

$$\begin{aligned} \mathbf{x}^{k+\frac{1}{2}} &= prox_\lambda(g)(\mathbf{x}^k - \tilde{\mathbf{x}}^k) \\ \mathbf{z}^{k+\frac{1}{2}} &= prox_\rho(f)(\mathbf{z}^k - \tilde{\mathbf{z}}^k) \\ (\mathbf{x}^{k+1}, \mathbf{z}^{k+1}) &= \Pi_{\mathbf{A}}(\mathbf{x}^{k+\frac{1}{2}} + \tilde{\mathbf{x}}^k, \mathbf{z}^{k+\frac{1}{2}} + \tilde{\mathbf{z}}^k) \\ \tilde{\mathbf{x}}^{k+1} &= \tilde{\mathbf{x}}^k + \mathbf{x}^{k+\frac{1}{2}} - \mathbf{x}^{k+1} \\ \tilde{\mathbf{z}}^{k+1} &= \tilde{\mathbf{z}}^k + \mathbf{z}^{k+\frac{1}{2}} - \mathbf{z}^{k+1}, \end{aligned} \tag{2.53}$$

where λ and ρ are relaxation parameters. The graph projection $\Pi_{\mathbf{A}}$ can be seen as a linear operator

$$\Pi_{\mathbf{A}}(\mathbf{c}, \mathbf{d}) = \begin{bmatrix} \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & -\mathbf{I} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{A}^T \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}, \tag{2.54}$$

where $\mathbf{I}$ is an unit matrix whose size is determined by the size of $\mathbf{A}$. $\mathbf{c}, \mathbf{d}$ are both vectors of the same size as $\mathbf{x}$ and $\mathbf{z}$.

If $f(\mathbf{z}) = \|\mathbf{z} - \mathbf{A}\mathbf{x}_{true}\|^2 = \|\mathbf{z} - \mathbf{y}\|^2$ and $g(\mathbf{x}) = 0$ or $\lambda TV(\mathbf{x})$, ADMM method solves Eq.1.5 without or with TV constraints respectively. When $g(\mathbf{x}) = 0$, the $\mathbf{x}^{k+\frac{1}{2}} = prox_\lambda(g)(\mathbf{x}^k - \tilde{\mathbf{x}}^k)$ in ADMM iteration can be written as

$$\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^k - \tilde{\mathbf{x}}^k. \tag{2.55}$$

After some manipulations, $\mathbf{z}^{k+\frac{1}{2}} = prox_\rho(f)(\mathbf{z}^k - \tilde{\mathbf{z}}^k)$ in the ADMM iteration can be written as

$$\mathbf{z}^{k+\frac{1}{2}} = \frac{1}{1+\frac{\rho}{2}}\mathbf{y} + \frac{1}{\frac{2}{\rho}+1}(\mathbf{z}^k - \tilde{\mathbf{z}}^k), \tag{2.56}$$

where $\mathbf{y}$ is the already obtained projection data.

Block ADMM assumes that both function $f(\mathbf{z})$ and $g(\mathbf{x})$ are block separable, i.e.

$$\begin{aligned} f(\mathbf{z}) &= \sum_{i=1}^{M} f_i(\mathbf{z}_{I_i}), \\ g(\mathbf{x}) &= \sum_{j=1}^{N} g_j(\mathbf{x}_{J_j}), \end{aligned} \tag{2.57}$$

where

$$\begin{aligned} \mathbf{z} &= [\mathbf{z}_{I_1}, \cdots, \mathbf{z}_{I_M}]^T \\ \mathbf{x} &= [\mathbf{x}_{J_1}, \cdots, \mathbf{x}_{J_N}]^T, \end{aligned} \tag{2.58}$$

with $\mathbf{z}_{I_i} \in \mathbb{R}^{m_i}$, $\mathbf{x}_{J_j} \in \mathbb{R}^{n_j}$, so $\sum_i^M m_i = r$ and $\sum_{j=1}^N n_j = c$. To apply block ADMM, MN new variables $\mathbf{x}_{J_j}^i \in \mathbb{R}^{n_j}$, $\mathbf{z}_{I_i}^j \in \mathbb{R}^{m_i}$ are introduced. Then the block ADMM, after a detailed deduction in (Parikh and Boyd, 2014), is expressed as

$$\begin{aligned} \mathbf{x}_{J_j}^{k+\frac{1}{2}} &= prox_{\lambda g_j}(\mathbf{x}_{J_j}^k - \tilde{\mathbf{x}}_{J_j}^k) \\ \mathbf{z}_{I_i}^{k+\frac{1}{2}} &= prox_{\rho f_i}(\mathbf{z}_{I_i}^k - \tilde{\mathbf{z}}_{I_i}^k) \\ (\mathbf{x}_{J_j}^{i\ k+\frac{1}{2}}, \mathbf{z}_{I_i}^{j\ k+\frac{1}{2}}) &= \Pi_{\mathbf{A}_{I_i}^{J_j}}(\mathbf{x}_{J_j}^k - (\tilde{\mathbf{x}}_{J_j}^i)^k, \mathbf{z}_{I_i}^{j\ k} + \tilde{\mathbf{z}}_{I_i}^k) \\ \mathbf{x}_{J_j}^{k+1} &= \mathbf{avg}(\mathbf{x}_{J_j}^{k+\frac{1}{2}}, \{\mathbf{x}_{J_j}^{i\ k+\frac{1}{2}}\}_{i=1}^M) \\ (\mathbf{z}_{I_i}^{k+1}, \{\mathbf{z}_{I_i}^{j\ k+1}\}_{j=1}^N) &= \mathbf{exch}(\mathbf{z}_{I_i}^{k+\frac{1}{2}}, \{\mathbf{z}_{I_i}^{j\ k+\frac{1}{2}}\}_{j=1}^N) \\ \tilde{\mathbf{x}}_{J_j}^{k+1} &= \tilde{\mathbf{x}}_{J_j}^k + \mathbf{x}_{J_j}^{k+\frac{1}{2}} - \mathbf{x}_{J_j}^{k+1} \\ \tilde{\mathbf{z}}_{I_i}^{k+1} &= \tilde{\mathbf{z}}_{I_i}^k + \mathbf{z}_{I_i}^{k+\frac{1}{2}} - \mathbf{z}_{I_i}^{k+1} \\ (\tilde{\mathbf{x}}_{J_j}^i)^{k+1} &= (\tilde{\mathbf{x}}_{J_j}^i)^k + (\mathbf{x}_{J_j}^i)^{k+\frac{1}{2}} - \mathbf{x}_{J_j}^{k+1}, \end{aligned} \tag{2.59}$$

where **avg** and **exch** are element-wise averaging and exchange operators, respectively. The exchange operator $\mathbf{exch}(c, \{c_j\}_{j=1}^N)$ is an operator that computes $\mathbf{z}_{I_i}$ based on $\{\mathbf{z}_{I_i}^j\}_{j=1}^N$ by reducing or adding a correction term, given by

$$\begin{aligned} \mathbf{z}_{I_i}^j &= c_j + \frac{(c - \sum_{j=1}^N c_j)}{N+1}, \\ \mathbf{z}_{I_i} &= c - \frac{(c - \sum_{j=1}^N c_j)}{N+1}. \end{aligned} \tag{2.60}$$

The final solution can be obtained from $[\mathbf{x}_1^{k+\frac{1}{2}}, \cdots, \mathbf{x}_N^{k+\frac{1}{2}}]$.

From the description, it can be seen that the most computationally intensive part of ADMM and its block form is the Π projection. For example, in ADMM, based on Eq.2.54, this projection is equivalent to solving the linear system

$$\begin{bmatrix} \mathbf{c} + \mathbf{A}^T\mathbf{d} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{c}} \\ \hat{\mathbf{d}} \end{bmatrix}, \tag{2.61}$$

where $\hat{\mathbf{c}}$ and $\hat{\mathbf{d}}$ are the projected $\mathbf{c}$ and $\mathbf{d}$. $\mathbf{A}$ can also be changed as $\mathbf{A}_I^J$ in block ADMM form. To solve this equation, the conjugate gradient for the least square problems (CGLS) is recommended due to its fast convergence rate (Parikh and Boyd, 2014). Each ADMM iteration requires several conjugate gradient (CG) iterations, leading to dozens or even hundreds of matrix-vector multiplies, which leads to significant computational cost per iteration. A standard technique to speed up the ADMM iteration is to terminate the CG iteration early. With a predefined tolerance parameter τ, the CG is stopped when the residual norm of Eq.2.61 is smaller than τ. It should be noted that the tolerance τ should influence the final image vector $\mathbf{x}_{rec}$'s accuracy. At low precision level, i.e. at the initial iteration stage, the influence to the $\mathbf{x}_{rec}^k$'s accuracy brought by τ is not significant so the τ is allowed to be set as a relative large value. When the iteration goes on, if one wish to obtain a high level accuracy solution of $\mathbf{x}_{rec}$, then the τ has to be tuned down and then the iteration computation cost increases. Early termination in the CG can result in more ADMM iterations, but leads to a much lower computational cost per iteration, giving an overall improvement in computational speed.

Apart from early termination, there are other methods to increase the ADMM computational efficiency. For example, a variable ρ-update scheme (Wang and Liao, 2001) can be adopted to accelerate the convergence rate.

The block ADMM shares its partition method on the system matrix $\mathbf{A}$ with our new methods. To the author's best knowledge, it is the only algorithm that enable each parallel node has partial access to the projection data $\mathbf{y}$ and reconstructed volume $\mathbf{x}$. As a result, similar to the previously mentioned SIRT-type methods, block ADMM will also be one of our main reference methods to be compared with the proposed methods. More detailed discussions and comparisons will be presented later.

2.5 Conclusions

For row-action methods, each 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 network, then each processor (or node) needs to store the whole reconstructed image vector $\mathbf{x}$ since each update is on the entire image, which can be computationally challenging, especially when performing the forward projection $\mathbf{A}_I\mathbf{x}$, back projection $\mathbf{A}_I^T(\mathbf{y} - \mathbf{A}\mathbf{x})$ and TV de-noising on $\mathbf{x}$. When the size of the reconstructed image continues to grow, the limited storage capability for each separate computation node (for example, GPUs in a multi-GPU distributed network) may lead to multiple data transmissions to cover the whole $\mathbf{x}$. 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

a small part of $\mathbf{x}$. However, they instead require full access to $\mathbf{y}$, which can again be prohibitive in large-scale situations. A combination of row and column-action methods, SBCD, is discussed in which each node only requires parts of the reconstructed image vector $\mathbf{x}$. However, this method requires the block residue $\mathbf{r}_I$ to be accurately calculated, implying that $\mathbf{x}$ still needs to be processed as a whole for each iteration.

In current CT system case, if the computation node can only process FP and BP involving a part of $\mathbf{x}$ and $\mathbf{y}$ and the TV de-noising on a part of $\mathbf{x}$, the requirement of a full access to either $\mathbf{y}$ or $\mathbf{x}$, which is the common feature of all widely-applied SIRT-type methods, leads to multiple communications between computation nodes and the master node (the node only stores variables but does not involve computations). This impedes the reconstruction speed. The only algorithm available prior to our work that overcomes these issues and allows computation nodes to operate with access to only part of $\mathbf{x}$ and $\mathbf{y}$ is the block ADMM method, whose original form has been applied in CT reconstruction. However, the required matrix inversions can become a limiting factors if these matrices are relatively large. A better combination of row and column-action methods to allow each iteration only address $\mathbf{y}_I$ and $\mathbf{x}_J$ is thus needed for CT reconstruction. Such approaches will be developed in this thesis.

Chapter 3

CSGD

The speed of the algorithm is empirically studied here by monitoring the change of the SNR of reconstructed image. Note that the inclusion of additional regularisation required to overcome the ill conditioning of the CT inverse problem is discussed in later chapters.

As mentioned in the previous chapter, when the size of the CT dataset, including the projection data vector $\mathbf{y}$ and reconstructed volume vector $\mathbf{x}$, have both exceed the storage capacity of the computation node, traditional SIRT-type CT reconstruction methods suffers from heavy communication cost due to the requirement of the full access to either $\mathbf{y}$ or $\mathbf{x}$. In this chapter, a novel algebraic reconstruction method called Coordinate-reduced Steepest Gradient Descent (CSGD) is presented in detail. The contributions of this chapter include: 1) This proposed algorithm is specially designed for the case where both projection data and reconstruction volume are so large that one computation node only has partial access to both. The communication overhead between computation nodes and the master node is reduced compared with the other SIRT-type methods because the parallel computation system has a tunable access to the dataset. Similar to the SBCD mentioned in section 2.2.3.4, CSGD selects a block of coordinates and estimates a stochastic gradient descent direction based on a row block of the system matrix $\mathbf{A}$. However, in CSGD, the residual $\mathbf{r}$ or $\mathbf{r}_I$ is not accurately calculated each time but is only partially updated. This method thus reduces the computation cost in each iteration and also enables a more flexible parallel application, at the cost of introducing a more stochastic update direction. Besides, the computation efficiency is much higher than block ADMM, which is the other main reference method apart from the SIRT-type methods. Simulations have verified the speed advantage of CSGD when compared with block ADMM. 2) The application details of the CSGD is explored in detail, including determining a rule of the parameter tuning to obtain the fastest reconstruction speed under a determined parallel network, researching algorithm's performance under different partitions on dataset $\mathbf{y}$ and $\mathbf{x}$, showing its ability to reconstruct the image into high

accuracy level under arbitrary dataset division, and proposing an importance sampling method to further accelerate the CSGD's reconstruction speed.

This chapter is widely based on the journal article "A Joint Row and Column Action Method for Cone-Beam Computed Tomography"(Gao and Blumensath, 2018b).

3.1 Algorithm description

3.1.1 Basic CSGD iteration

The proposed CSGD is inspired by SBCD discussed in the previous chapter, which is modified to 1) find a better strategy to compute step-length μ and 2) to efficiently approximate the residue $\mathbf{r}_I$.

After selecting the row block I, CSGD turns to optimize a block of objective function

$$f_I(\mathbf{x}) = \frac{1}{2}(\mathbf{y}_I - \mathbf{A}_I\mathbf{x})^T(\mathbf{y}_I - \mathbf{A}_I\mathbf{x}). \tag{3.1}$$

The gradient is

$$\mathbf{g} = \nabla f_I(\mathbf{x}) = (\mathbf{A}_I)^T(\mathbf{y}_I - \mathbf{A}_I\mathbf{x}) = (\mathbf{A}_I)^T\mathbf{r}_I. \tag{3.2}$$

If $\mathbf{x}$ moves along this direction, all elements in $\mathbf{x}$ are changed. Instead, we here use a coordinate descent approach in which only those voxel elements are updated whose indices are in the set J. The descent direction is then

$$\widetilde{\mathbf{g}} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{g}_J \\ \vdots \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ (\mathbf{A}_I^J)^T\mathbf{r}_I \\ \vdots \\ \mathbf{0} \end{bmatrix}. \tag{3.3}$$

It is worthy to mention that Eq.3.3 assumes that the J contains sequential column indexes. It is for the purpose of simplifying the expression of the equation and it is not a necessary requirement in the partition of $\mathbf{x}$ space.

Along with this new modified direction, the update, from the current k^{th} iteration to the next $k+1^{th}$ iteration, becomes

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mu\widetilde{\mathbf{g}} \rightarrow \begin{cases} \mathbf{x}_J^{k+1} & = \mathbf{x}_J^k + \mu\mathbf{g}_J \\ \mathbf{x}_{\widehat{J}}^{k+1} & = \mathbf{x}_{\widehat{J}}^k, \end{cases} \tag{3.4}$$

where μ is the step-length and $\widehat{J}$ is the complement to the set J. To compute the optimal μ that would lead to steepest descent, the direction of the gradient of $\mathbf{x}$ in the

next iteration ($\nabla f_I(\mathbf{x}^{k+1})$) should be perpendicular to the current update direction ($\widetilde{\mathbf{g}}$) (Wang, 2008), i.e.

$$((\mathbf{A}_I)^T(\mathbf{y}_I - \mathbf{A}_I\mathbf{x}^{k+1}))^T\widetilde{\mathbf{g}} = 0. \tag{3.5}$$

Using the fact that $\mathbf{x}^{k+1} = \mathbf{x}^k + \mu\widetilde{\mathbf{g}}$ and $\mathbf{r}_I = \mathbf{y}_I - \mathbf{A}_I\mathbf{x}^k$, which means the current partial residual, Eq.3.5 can be expanded as:

$$(\mathbf{r}_I)^T\mathbf{A}_I\widetilde{\mathbf{g}} = \mu\widetilde{\mathbf{g}}^T(\mathbf{A}_I)^T\mathbf{A}_I\widetilde{\mathbf{g}}. \tag{3.6}$$

Attention that $(\mathbf{r}_I)^T\mathbf{A}_I\widetilde{\mathbf{g}} = \widetilde{\mathbf{g}}^T\mathbf{A}_I^T\mathbf{r}_I = \widetilde{\mathbf{g}}^T\mathbf{g} = \mathbf{g}_J{}^T\mathbf{g}_J$, where the last equation is obtained by the sparse property of $\widetilde{\mathbf{g}}$. Besides, using this sparse property, it is also easy to obtain $\mathbf{A}_I\widetilde{\mathbf{g}} = \mathbf{A}_I^J\mathbf{g}_J$. Eq.3.6 is then expressed as:

$$\mu = \frac{\mathbf{g}_J{}^T\mathbf{g}_J}{\mathbf{g}_J{}^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J}. \tag{3.7}$$

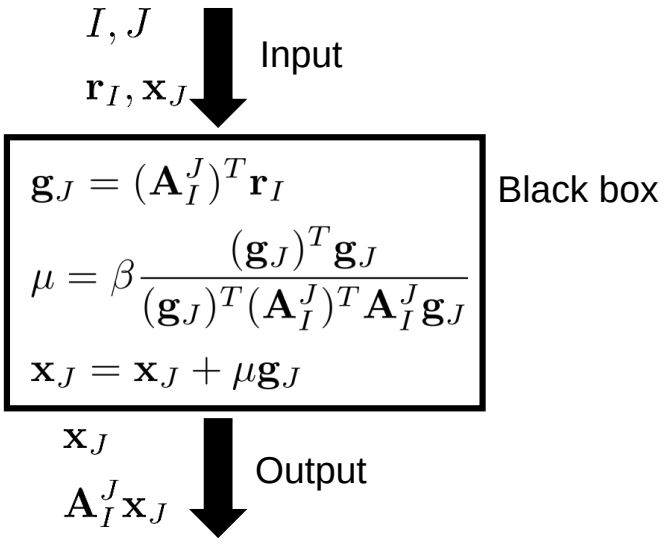
This calculation does not require a computation node to have access to the whole row or column block of matrix $\mathbf{A}$. When the matrix $\mathbf{A}$ cannot be stored and needs to be generated on the fly, CSGD iterations thus only require computations with the submatrix $\mathbf{A}_I^J$ and its transpose.

An important issue is that the step size derived from Eq.3.7 minimises the current $\|\mathbf{r}_I\|$ instead of $\|\mathbf{r}\|$. 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 and is not guaranteed to reduce $\|\mathbf{r}\|$. Furthermore, it is not guaranteed that our update direction $\widetilde{\mathbf{g}}$ is always a descending direction of the original cost function. To stabilise our algorithm, we thus introduce an additional relaxation parameter β, which is a constant between 0 and 1 and needs to be manually tuned in application, into the calculation of μ. This helps us to avoid overshooting the minimum if $\widetilde{\mathbf{g}}$ is a descent direction, whilst in cases in which $\widetilde{\mathbf{g}}$ is not a descent direction, the increase in $\mathbf{r}$ remains small.

The basic iteration of CSGD can be viewed as being performed in a block box and the input are $\mathbf{r}_I, \mathbf{x}_J$ and the output is $\mathbf{x}_J$, as is illustrated in Fig.3.1. The output not only includes the updated image block $\mathbf{x}_J$ but also includes the FP result of $\mathbf{x}_J$, which will be used to update the residual $\mathbf{r}$ and will be explained in the next section.

3.1.2 CSGD algorithm

Fig.3.1 shows that CSGD iteration is especially suitable for the case where every computation node, which plays as the block box role, has partial access to both $\mathbf{y}$ and $\mathbf{x}$. It can be seen that the update on $\mathbf{r}_I$ plays an important role in updating $\mathbf{x}$. As a result, how to efficiently update $\mathbf{r}_I$, where $I \in \{I_i\}_{i=1}^M$ is the next problem that CSGD tries to



(a) SNR

Figure 3.1: The basic CSGD iteration can be viewed as being sealed in a black box. Such black box can be a GPU or a computer in a parallel network.

solve. Mathematically, $\mathbf{r}_I$ can be obtained by calculating $\mathbf{y}_I - \mathbf{A}_I\mathbf{x} = \mathbf{y} - \sum_{j=1}^{N} \mathbf{A}_I^{J_j}\mathbf{x}_{J_j}$. This means that for each update of $\mathbf{r}$, we require N matrix-vector multiplications (i.e. $\mathbf{A}_I^{J_j}\mathbf{x}_{J_j}$). This violates the original desire to reduce the computations and communication overhead in each iteration. For example, if the number of parallel nodes in a network is p and $p = \frac{N}{2}$, then the accurate calculation of $\mathbf{r}_I$ (or the accurate calculation of $\mathbf{A}_I\mathbf{x}$) requires each parallel node to communicate with the master node twice, whilst the expected communication time is 1. One possible solution to approximate $\mathbf{A}_I\mathbf{x}$ is to only sum up the results of recently computed matrix-vector products $\mathbf{A}_I^J\mathbf{x}_J$, which can be computed and outputted by each parallel node, as indicated in Algo.3.1. It should be noted that in Algo.3.1, the α and γ are percentages of selected row and column blocks. They cannot be set as arbitrary numbers but should make αM and γN as integrals between $[1, M]$ and $[1, N]$ respectively. In a parallel network with p parallel computation nodes, it is suggested to set α, γ to make $\alpha\gamma MN = p$, making each computation node receive different $\{\mathbf{r}_I, \mathbf{x}_J\}$ pairs.

It can be seen that Algo.3.1 is parallelizable on the J-loop (line 7). To be more specific, line 8 to 12 are performed in each computation node in parallel, as shown in Fig.3.1. The line 13 illustrates that the $\widehat{\mathbf{y}_I}$, which is stored on the master node and is cleared at the beginning of each inner iteration, is used to collect $\mathbf{z}_I^j$ generated by each parallel node and sum them up for the future's update on $\mathbf{r}_I$. Here the master node is a node that is responsible for communicating with all parallel computation nodes. A detailed discussion on the communication between master node and the computation nodes will be presented in Chapter 6. This update scheme on $\mathbf{r}_I$ seems appealing since it greatly reduce the computation amount in the update scheme. For example, if the number of

Algorithm 3.1 Preliminary algorithm for CSGD

1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\widehat{\mathbf{y}} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. $\alpha \in (0, 1]$, $\gamma \in (0, 1]$ are the percentage of the selected row and column blocks.
2: **for** $epoch = 1, 2, \cdots, K_{max}$ **do**
3: **for** $ii = 1, 2, \cdots, \alpha M$ **do**
4: Random select a row block I from sets $\{I_i\}_{i\in[1,M]}$ with replacement
5: $\mathbf{r}_I = \mathbf{y}_I - \widehat{\mathbf{y}_I}$
6: $\widehat{\mathbf{y}_I} = \mathbf{0}$
7: **for** $jj = 1, 2, \cdots, \gamma N$ **do**
8: Random select a row block J from sets $\{J_j\}_{i\in[1,N]}$ with replacement, record the index j as well
9: $\mathbf{g}_J = (\mathbf{A}_I^J)^T \mathbf{r}_I$
10: $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}_I^J)^T \mathbf{A}_I^J \mathbf{g}_J}$
11: $\mathbf{x}_J = \mathbf{x}_J + \mu \mathbf{g}_J$
12: $\mathbf{z}_I^j = \mathbf{A}_I^J \mathbf{x}_J$
13: $\widehat{\mathbf{y}_I} = \widehat{\mathbf{y}_I} + \mathbf{z}_I^j$
14: **end for**
15: **end for**
16: **end for**
17: $\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

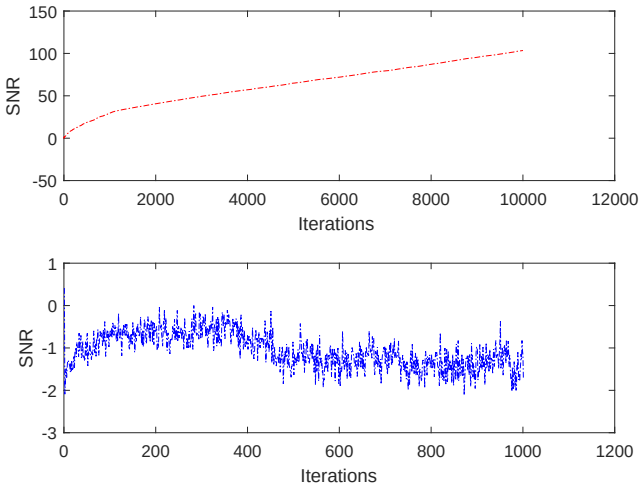
parallel nodes $p = \frac{N}{2}$, then the computation amount of the update on $\mathbf{r}_I$ is halved. However, simulation results indicate that this algorithm does not approach the least squares solution because of those missing $\mathbf{z}_I^j$ which should also engage in the update of $\mathbf{r}_I$. As a result, the efficient updating $\mathbf{r}_I$ in CSGD needs to satisfy two requirements: 1) the update does not need to calculate $\mathbf{z}_I^j = \mathbf{A}_I^{J_j} \mathbf{x}_{J_j}$ to cover the whole $\{J_j\}_{j=1}^N$. 2) the update cannot neglect those missing $\mathbf{z}_I^j$ that are not output by current parallel nodes. One solution to meet these two requirements is to make the master node record all $\mathbf{z}_I^j$ and gradually update them by p parallel nodes, thus when updating $\mathbf{r}_I$, there are p $\mathbf{z}_I^j$ that contain the latest information whilst the other $N - p$ $\mathbf{z}_I^j$ contains the outdated information. The algorithm is indicated in Algo.3.2. Line 10 in Algo.3.2 means that only a small fraction of $\mathbf{z}_I^j$ is updated by the latest $\mathbf{x}_J$ whilst the other $\mathbf{z}_I^j$ remains unchanged. However when updating $\mathbf{r}_I$ or $\mathbf{r}$ in line 12, all $\{\mathbf{z}_I^j\}_{j=1}^N$ engage despite some of them store stale information.

Despite that the Algo.3.1 does not generate iterations that approach the least squares solution, it provides a framework of CSGD algorithm and Algo.3.2, which is a version of CSGD algorithm that can empirically be shown to approach the least squares solution, is a refined version of Algo.3.1. To verify Algo.3.2 has the ability to solve the noise free linear inverse problem whilst Algo.3.1 does not, a random matrix $\mathbf{A} \in \mathbb{R}^{200\times100}$ was generated together with a vector $\mathbf{x} \in \mathbb{R}^{100\times1}$. All elements in $\mathbf{A}$ and $\mathbf{x}$ are uniformly distributed random numbers in the interval (0,1). The projection $\mathbf{y}$ was then $\mathbf{y} = \mathbf{A}\mathbf{x}$. The matrix was divided into 4 row blocks and 4 column blocks using consecutive

Algorithm 3.2 CSGD algorithm which parallelizes the column blocks (called as CSGD(J)).

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α, γ are the percentage of the selected row and column blocks.
for $epoch = 1, 2, \cdots, K_{max}$ **do**
  for $jj = 1, 2, \cdots, \gamma N$ **In parallel** (J loop) **do**
    Random select a column block J from sets $\{J_j\}_{j\in[1,N]}$ with replacement and record the index number j
    for $ii = 1, 2, \cdots, \alpha M$ **In sequential** (I loop) **do**
      Random select a row block I from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      $\mathbf{g}_J = (\mathbf{A}_I^J)^T \mathbf{r}_I$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}_I^J)^T \mathbf{A}_I^J \mathbf{g}_J}$
      $\mathbf{x}_J = \mathbf{x}_J + \mu \mathbf{g}_J$
      $\mathbf{z}_I^j = \mathbf{A}_I^J \mathbf{x}_J$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N} \mathbf{z}^j$
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

rows/columns. Results indicated in Fig.3.2 demonstrate the importance as well as the effectiveness of recording all $\mathbf{z}_I^j$ and partially update some of them.



(a) SNR

Figure 3.2: The y-axis is the SNR (defined in Eq.1.7) of the reconstructed image vector, the unit is dB. β is 0.2. The difference is significant when using Algo.3.1 (the blue line) and using Algo.3.2 (the red line). It clearly indicates that it is necessary to use stale $\mathbf{z}_I^j$ to update $\mathbf{r}_I$ otherwise the updated $\mathbf{r}_I$ is not a "good enough" estimation of the real $\mathbf{y}_I - \mathbf{A}_I\mathbf{x}$ and thus leads to false update directions as well as step-lengths.

Algo.3.2 is inherently parallelisable only over the volume block J and the maximum number of parallel nodes is γN ($\gamma \in (0, 1]$). For the I-loop, it is still sequential. In a centralised network, a general work scheme for CSGD is illustrated in Fig.3.3. In this scheme, take the right computation node as an example, this computation node receives the reconstructed image block $\mathbf{x}_{J_4}$ obtained from the last iteration. It also receives a row block index set I_2 used to calculate $(\mathbf{A}_{I_2}^{J_4})^T\mathbf{r}_{I_2}$ and $\mathbf{A}_{I_2}^{J_4}(\mathbf{A}_{I_2}^{J_4})^T\mathbf{r}_{I_2}$. After the update of $\mathbf{x}_{J_4}$ and $\mathbf{z}_{I_2}^4$, the node makes a request to the master node and obtains another row block index set I_3 and repeats the process. When the two processes are finished, the values of $\mathbf{z}_{I_2}^4$ and $\mathbf{z}_{I_3}^4$ are returned to the master node which performs the update on $\mathbf{r}$. Further analysis indicates that this method can be further parallelized over both row and column blocks. This algorithm is indicated in Algo.3.3.

Using the "master-servant" parallel model, the architecture of CSGD(I,J) is indicated in Fig.3.4. The maximum number of parallel computation nodes increases from γN to $\alpha\gamma MN$ and different row blocks I for the same column block $\mathbf{x}_J$ can be performed in parallel, thus reducing the computation load for each computation node.

3.2 Preliminary simulations

Before the discussion on the CSGD property, it is worthy to mention that it is hard to analysis CSGD mathematically. This is because that the CSGD uses highly stochastic

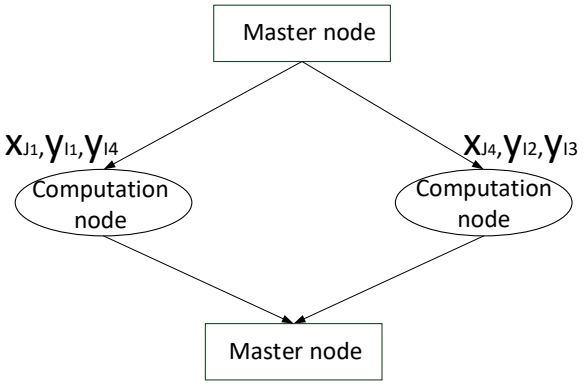


Figure 3.3: The general work scheme for CSGD(J). The master node randomly assigns several row and column blocks to several computation nodes. The current iteration only updates $\mathbf{x}_{J_1}$ with $\mathbf{y}_{I_1}$,$\mathbf{y}_{I_4}$, and $\mathbf{x}_{J_4}$ with $\mathbf{y}_{I_2}$,$\mathbf{y}_{I_3}$. To meet the parallel computation condition, the column blocks J assigned to different computation nodes should not overlap with each other. Within each computation node, the update on different row blocks I cannot be further parallelised and thus the iteration is performed in a sequential form, which limits the scalability of the proposed algorithm.

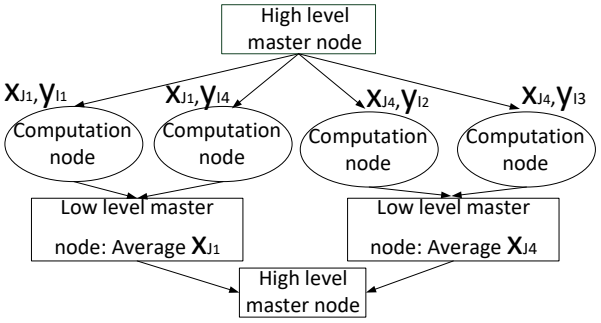


Figure 3.4: The general work scheme for CSGD(I,J). The current iteration only updates $\mathbf{x}_{J_1}$ with $\mathbf{y}_{I_1}$,$\mathbf{y}_{I_4}$, and $\mathbf{x}_{J_4}$ with $\mathbf{y}_{I_2}$,$\mathbf{y}_{I_3}$. The high level master node is only responsible for collecting reconstructed image block, updating residue $\mathbf{r}$ and assigning projection data block and image block obtained from the last iteration. The low level master nodes average each reconstructed block, while computation nodes are main nodes calculating the CSGD iteration.

update direction in each iteration, and the direction is not an unbiased estimation of the gradient. This means the traditional convergence analysis of SGD is not appropriate for CSGD. We thus here do not derive a mathematical convergence proof but instead use empirical performance validation of CSGD.

To reflect the CSGD performance, the SNR of the reconstructed image vector is used to reflect the reconstruction quality, which is defined in Eq.1.7. In the following, the term "reconstruction speed" will be frequently used as one measurement of the algorithm's property. This term reflects the change of SNR along with realistic spending time. Furthermore, the usage times of matrix-vector multiplications (i.e. the forward projection

Algorithm 3.3 CSGD algorithm which parallelizes both row and column blocks (called CSGD(I,J)).

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
for $epoch = 1, 2, \cdots, K_{max}$ **do**
  $\hat{\mathbf{x}} = \mathbf{0}$
  for jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
    Random select a column block J from sets $\{J_j\}_{j\in[1,N]}$ with replacement and record the index number j
    for ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
      Random select a row block I from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      $\mathbf{g}_J = (\mathbf{A}_I^J)^T\mathbf{r}_I$
      $\mu = \beta\frac{(\mathbf{g}_J)^T\mathbf{g}_J}{(\mathbf{g}_J)^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J}$
      $\mathbf{z}_I^j = \mathbf{A}_I^J(\mathbf{x}_J + \mu\mathbf{g}_J)$
      $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu\mathbf{g}_J$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N}\mathbf{z}^j$
  For all blocks J that have been updated, $\mathbf{x}_J = \hat{\mathbf{x}}_J$/(the times of block J has been updated)
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

$\mathbf{A}_I^J\mathbf{x}_J$ as well as $\mathbf{A}_I^J\mathbf{g}_J$ and the backward projection $(\mathbf{A}_I^J)^T\mathbf{r}_I$) can be used as a reference of the actual spending time. This is because the matrix-vector multiplication is the most time-consuming part in each iteration. A brief simulation will present later to further prove this claim.

In this section, CSGD is applied to both 2D fan-beam and 3D cone-beam CT system with circular scanning trajectories. Simulations demonstrate that the acceptable range of β leading to high accuracy solutions and the reconstruction speed change relative to the percentages of row and column blocks used in each iteration. Since block ADMM has the same parallel computation architecture with CSGD and also allows each computation node has partial access to the dataset $\mathbf{y}$ and $\mathbf{x}$, it is used as a main reference method in this section to compare with CSGD. Unless stated otherwise, the partition method uses consecutive rows and columns. A detailed discussions on differences between using sequential or shuffled row/column indexes of $\mathbf{y}$ and $\mathbf{x}$ is presented in section 3.2.3.

The 2D fan-beam CT scanning geometry is indicated in Fig.3.5. The scanned image adopts a discretized $K \times K$ pixels phantom image provided inherently in Matlab. The elements outside the $K \times K$ box are all 0. The point source P starts from the above of the scanned object and the source-detector pair rotates around the object for a circular circle. The projection is measured with a fixed rotation interval. The linear detector

contains several evenly distributed detectors. The rays connecting the point source P and these pixels are then measured. When generating the projection data, it uses the same FP model as that used in the reconstruction. In other words, despite that original image is continuous, it is first discretized into $K \times K$ pixels and then is scanned to generate the projection $\mathbf{y}$. When evaluating the quality of the reconstructed image $\mathbf{x}_{rec}$, the reference is also the discretized $K \times K$ scanned image vector.

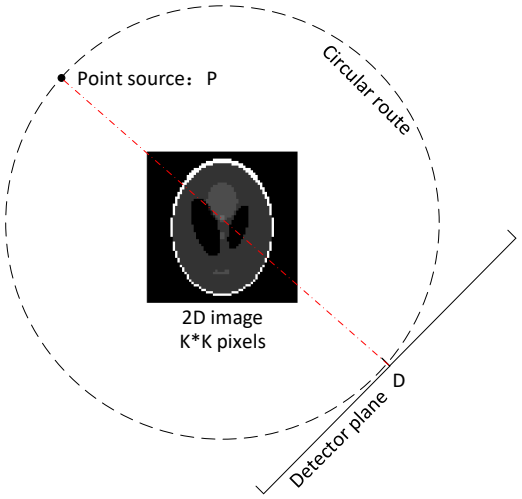


Figure 3.5: The scanning geometry, where the P is the point source location, O is the geometry center of the object and the rotation center. The D is the geometry center of the detector. The POD three points are always on one line and is always perpendicular to the detector.

The first simulation using Fig.3.5 is to verify that the usage of $\mathbf{A}_I^J$ is able to reflect the actual spending time as the computations corresponding to the $\mathbf{A}_I^J$ is the most time-consuming part among the iterations. A large CT scanning geometry is used: K in Fig.3.5 is 1204, $OP = OD = 2000$. The rotation interval is 0.1°. The detector contains 4000 pixels. Then corresponding $\mathbf{A} \in \mathbb{R}^{(1.44\times10^7)\times(1.0\times10^6)}$. When $N = 4, \alpha = \frac{1}{M}, \gamma = 0.5$, the percentage of each operation takes is illustrated in Table 3.1. M is changed from 2 to 200. Since N is fixed, increasing M means that the size of $\mathbf{A}_I^J$ decreases and thus the computation cost for the matrix-vector multiplication is reduced. Despite this, it can be seen that even when M is increased to a large number (200), the three matrix-vector multiplications are still the most time consuming parts, taking 94% of the whole time. This proves that using the "usage of $\mathbf{A}_I^J$" to reflect the actual computation time is reasonable.

In the following simulations, unless stated otherwise, K in Fig.3.5 is 16 and the size of pixels on the detector and on the image are both 1, the length of OP and OD are always 100. The rotation interval for point source as well as the detector is 10°. The detector contains 30 pixels. As a result, the system matrix $\mathbf{A}$, whose sparsity is similar to the large-scale CT reconstruction cases, only contains $30 \times \frac{360}{10} = 1080$ rows and

Table 3.1: Matlab profiler results of each CSGD operation

M=2		M=200	
operations	percentage(%)	operations	percentage(%)
$(\mathbf{A}_I^J)^T\mathbf{r}_I$	33.2	$(\mathbf{A}_I^J)^T\mathbf{r}_I$	31.5
$\mathbf{A}_I^J\mathbf{g}_J$	32.9	$\mathbf{A}_I^J\mathbf{g}_J$	31.4
$\mathbf{A}_I^J\mathbf{x}_J$	32.6	$\mathbf{A}_I^J\mathbf{x}_J$	31.0
other line	1.3	other line	6.1

$16 \times 16 = 256$ columns. The reason why a small-scale 2D scanning problem is used here is that it can reduce the time for simulation, enabling to perform more simulations to exploit various properties of CSGD and to test the performance of CSGD after long enough iterations. It can reflect the performances of CSGD in the large scale situation because of two reasons. The first reason is that the system matrices in the small-scale and in the large-scale situation are similar to each other and both of them are very sparse. To verify it, an intensity distribution of system matrix $\mathbf{A}$ in the small-scale case is indicated in Fig.3.6. The matrix is treated as an image and the different colours

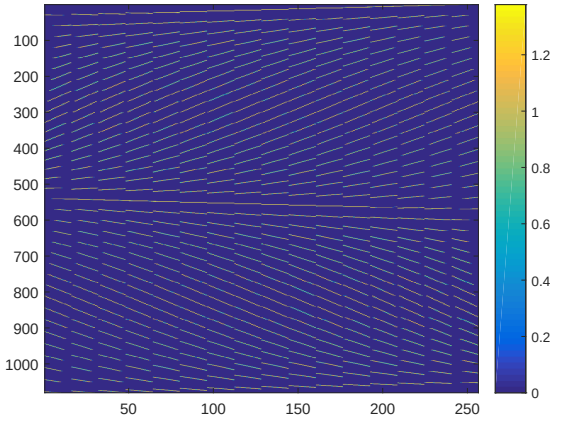
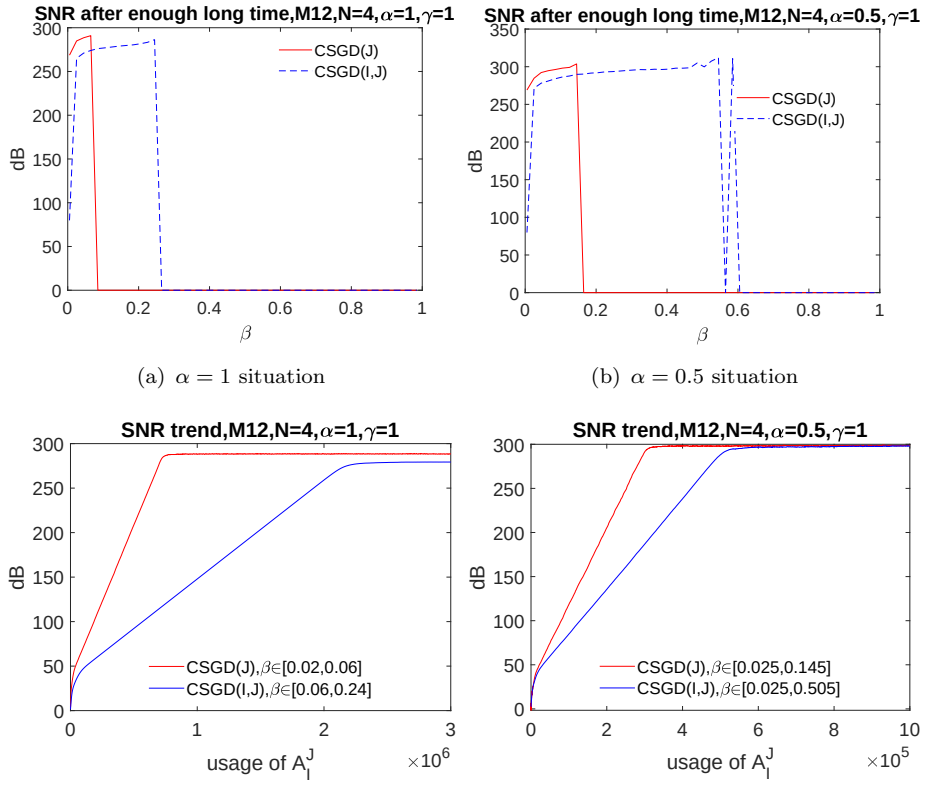


Figure 3.6: The intensity distribution of matrix $\mathbf{A} \in \mathbb{R}^{1080\times256}$ in the tiny CT system. The horizontal direction is the column index and vertical direction is the row index of $\mathbf{A}$. Different colours reflect different element's values, as shown in the right side bar. For a certain X-ray, it only passes though a few image pixels. As a result, the system matrix is very sparse. This property arouses the following importance sampling strategy.

present different pixel values. It can be seen that the majority elements in $\mathbf{A}$ is 0 and the non-zero element actually takes less than 1% among the whole elements. This property widely exists in all CT-related system matrices regardless of the size of them. The second reason is that the $\mathbf{y}, \mathbf{A}, \mathbf{x}$, although is not of enormous size, are still divided into blocks and the iteration only uses partial of them. This is the same with the application in the

large-scale dataset. The simulation results are thus able to provide useful reference to predict the large scale properties.

A noise free simulation was conducted to verify the difference between CSGD(J) and CSGD(I,J) as well as their abilities to approach to the true solution. In this noise free simulation, all data are stored in double-precision floating-point format and M=12,N=4, $\alpha = 1$ or 0.5. The SNR of the reconstructed image $\mathbf{x}_{rec}$ is shown in Fig.3.7. Fig.3.7 (a)



(a) $\alpha = 1$ situation (b) $\alpha = 0.5$ situation

(c) $\alpha = 1$ situation (d) $\alpha = 0.5$ situation

Figure 3.7: In this chapter, all simulations are repeated with different β within range (0, 1] and the fastest reconstruction speed is presented. (a) and (b) are the final SNR of reconstructed image vectors. For the purpose of visualisation, all the negative SNR, which means divergent iteration due to improper β, are set to zeros. (c) and (d) are the fastest reconstruction speeds of two CSGD algorithms selected from proper range of β.

and (b) has shown that CSGD(J) and CSGD(I,J) can both approach to the true solution $\mathbf{x}_{true}$ closely when the system is overdetermined and is noise-free. This is because that within a proper range of β, the SNR of the final results is about 300dB, which is almost the precision limit of the double precision floating-point format operation. According to the definition of SNR, this means that the final iterated $\mathbf{x}_{rec}$ is close enough to the $\mathbf{x}_{true}$. Fig.3.7 (c) and (d) show that the CSGD(I,J) method is slow, but it has a more extensive range of β that leads to highly accurate solutions. This means that the parameter tuning is easier than for CSGD(J). Furthermore, CSGD(I,J) also enables more computation nodes to engage in the iteration, thus if the parallel network has sufficient computation

node, the CSGD(I,J) is more suitable than CSGD(J). For example, in Fig.3.7(d) case, when $\alpha = 0.5, \gamma = 1$, CSGD(I,J) allows for $0.5 \times 12 \times 4 = 24$ computation nodes to compute CSGD basic iteration in parallel whilst CSGD(J) only allows 4 nodes to be engaged in the parallel iteration. If one usage of $\mathbf{A}_I^J$ spent δt seconds and the number of parallel nodes in a network is p, the actual spent wall time can be approximated as $\delta t \times$ usage of $\mathbf{A}_I^J/p$. This is because that the parallel node performs calculation simultaneously and within δt wall time, there are p matrix-vector multiplications to be performed. So more engaged parallel node means less actual wall time for the same usage of $\mathbf{A}_I^J$. If blue line in Fig.3.7(d) is divided by 24 and red line is divided by 4, then the trend approximately reflects the reconstruction speed over the actual wall time and then it will show CSGD(I,J) is faster than CSGD(J). As a result, the following simulations mainly focus on CSGD(I,J) despite its slower reconstruction speed.

3.2.1 Range of β for different M,N and α, γ

This part first discusses the rough range of β for which the algorithm is able to reach the true solution in the noise free setting, by setting M, N as different values. For simplicity, α and γ in this section are always 1, i.e. the number of parallel nodes is MN. In the following content of the thesis, unless specially stated in the figure, the projection data $\mathbf{y}$ are noise free and all variables are stored as single-precision floating-point format to reduce the simulation time. Experiments have verified that using single-precision floating-point format makes the SNR limit decreases from around 300dB to 100dB. This can be verified by performing long enough GD iterations and then observe the SNR limit. The simulation results are not presented here.

The SNR after 2000 epochs (simulations show that it is long enough to reach the true result) for different β is indicated in Fig.3.8.

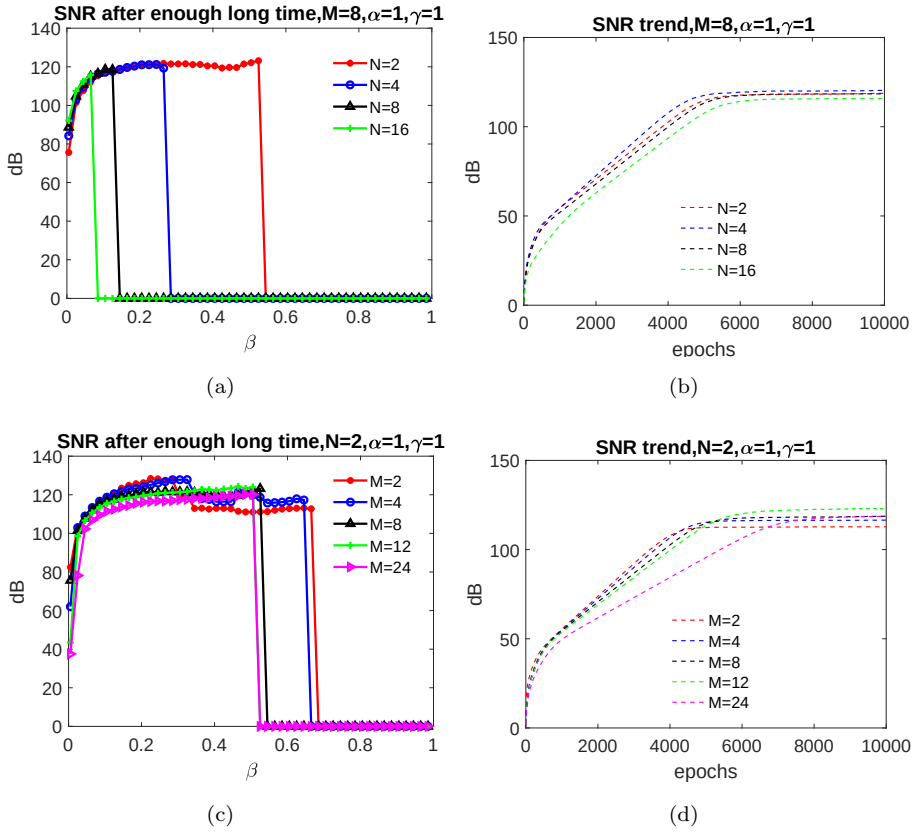


Figure 3.8: The available range of β is influenced more by N instead of M. When N increases, the range of β shrinks rapidly. Different choice of M, N, β lead to different final precision level but this property is not important. This is because in CT reconstruction area, a medium accuracy solution (around 20 dB) is already clear enough to present all inner details of the scanned object and thus only initial reconstruction speed is of interest here.

It indicates that when N is fixed, increasing M gradually narrows the range of β. When M is fixed, increasing N rapidly narrows the range of β. In both simulations, CSGD presents a good slicing scalability. Here "epochs" is used instead of "usage of $\mathbf{A}_I^J$" to compare the speed under different partitions. This is because different M, N lead to different computation costs per projection, whilst $\alpha = \gamma = 1$ ensures that each epoch covers all sub-matrices $\mathbf{A}_I^J$ and thus have the same computation cost. Simulations are also conducted when the linear system has added noise, i.e. $\mathbf{e}$ in Eq.1.6 is not zero. In the following simulations, $\mathbf{e}$ is a Gaussian noise whose average value is zero and variance is σ. The generation of such type $\mathbf{e}$ in MATLAB is easy to be achieved by using function $randn()$. The above simulation was repeated for several times when $\sigma = 0.01$ (Fig.3.9) and the $\mathbf{e}$ is independently generated form the function $randn()$ each time. The simulation results are similar to each other, here only present one result from all simulations. Generally, the noisy and noise free models have the same properties. They can all reconstruct the image to a high accuracy and the reconstruction speed is almost the same. Here it is worthy to mention that under the noisy case, the semi-convergence

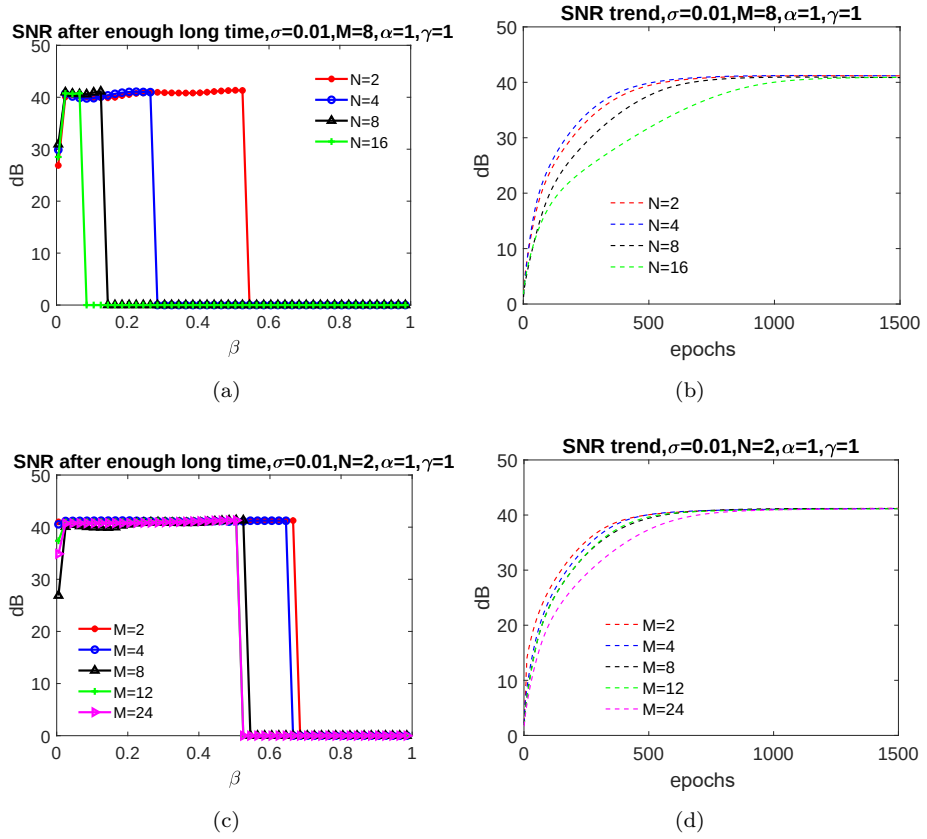


Figure 3.9: When $\sigma = 0.01$, the SNR of $\mathbf{y}$ is 100 dB. The range of β is similar to the above noise free model. The final SNR all comes to the same precision level. When M or N increases, the reconstruction speed decreases.

is existing and this phenomena is significant if the noise are severe. However, in the simulations shown in Fig.3.9, the semi-convergence phenomena is not significant, this is because of two reasons. The first reason is the scanning model is a 2D model, and thus the formed linear model is not severely ill-posed (Natterer, 2001). The second reason is that the added noise is not severe. These two reasons makes the semi-convergence phenomena not obvious. Simulations have shown that if the σ is increased from 0.01 to 0.1, then the semi-convergence phenomenon becomes significant but the final SNR also comes to the same precision level as shown in Fig.3.9. As a result, it is not presented in this part.

For a specific partition, when the number of parallel nodes (defined as p in this thesis) is smaller than MN, setting α and γ for fast reconstruction speed is another aspect worth exploring. Two noise-free cases are tested: 1)short-fat (SF) case: M=12, N=2 with $\gamma = 1$ and $\gamma = 1/2$ case are indicated in Fig.3.10. It can be seen that when γ is determined, reducing the α accelerate the reconstruction speed. This is because that a smaller α means more frequent update on $\mathbf{x}$ space within the same usage of $\mathbf{A}_I^J$ and the frequent update on $\mathbf{x}$ has compensate the slow reconstruction speed due to the stochastic update direction. When $\alpha = \frac{1}{12}$, a comparison for different γ is indicated in

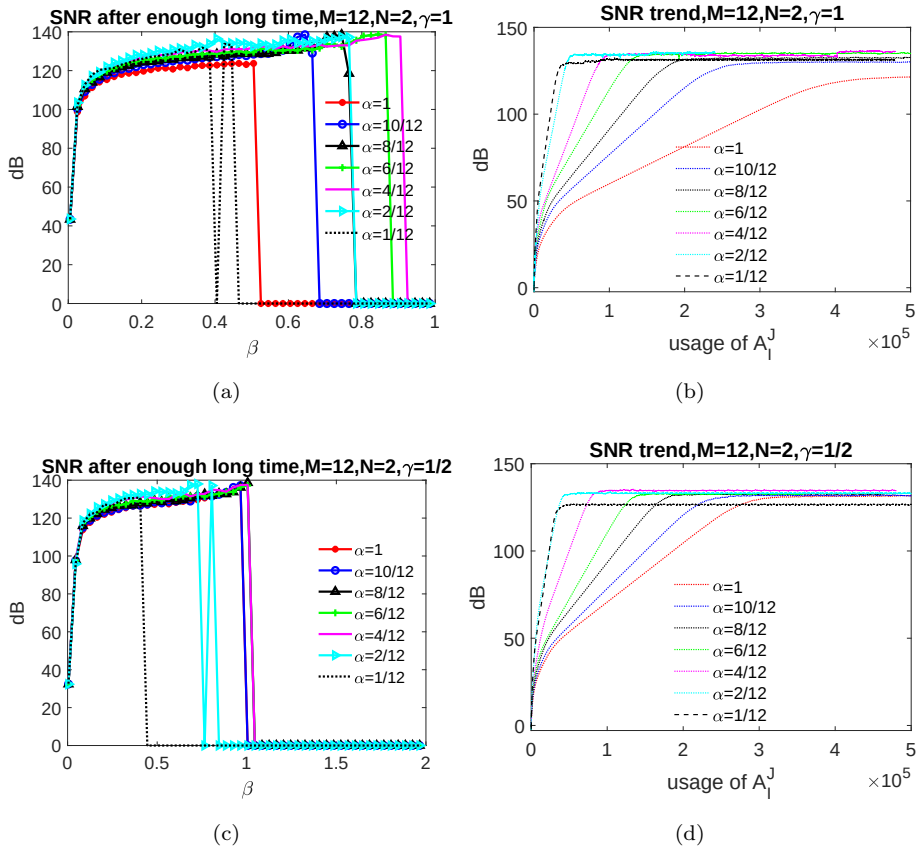


Figure 3.10: For $M = 12, N = 2$ SF situation, reducing α enlarges the range of acceptable β, except the situation when $\alpha = \frac{1}{M}$, where the CSGD algorithm becomes unstable within a narrow β range. A smaller α also increases the reconstruction speed than $\alpha = 1$ situation.

Fig.3.11. It can be seen that when $\alpha = \frac{1}{M}$, reducing γ does not significantly influence the reconstruction speed. This is because that each column block $\mathbf{x}_J$ is independently updated and has the same update frequency. Despite that a smaller γ makes some $\mathbf{z}_I^j$ outdated and thus cause the update $\mathbf{r}_I$ inaccurate, the influence is not severe because in

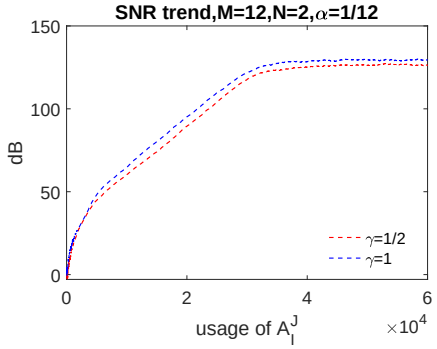


Figure 3.11: $\alpha = \frac{1}{M}, \gamma = \frac{1}{N}$ is not significantly slower than $\gamma = 1$ case, which suggests that the CSGD allows to reduce γ while maintaining reconstruction speed. This property can be useful when p is smaller than N.

reality partition, the N is usually not large (in this thesis, the N is usually set as 4,8,16 whilst M can reach 60), thus minimising γ into minimal $\frac{1}{N}$ and random select $\mathbf{x}_J$ does not cause severe delay. 2) The trend in Fig.3.10 and Fig.3.11 can be repeated when the shape of $\mathbf{A}_I^J$ is tall-thin (TT) case (i.e M=N=8), the simulation results are indicated in Fig.3.12. It again demonstrate that the shape does not influence reconstruction speed

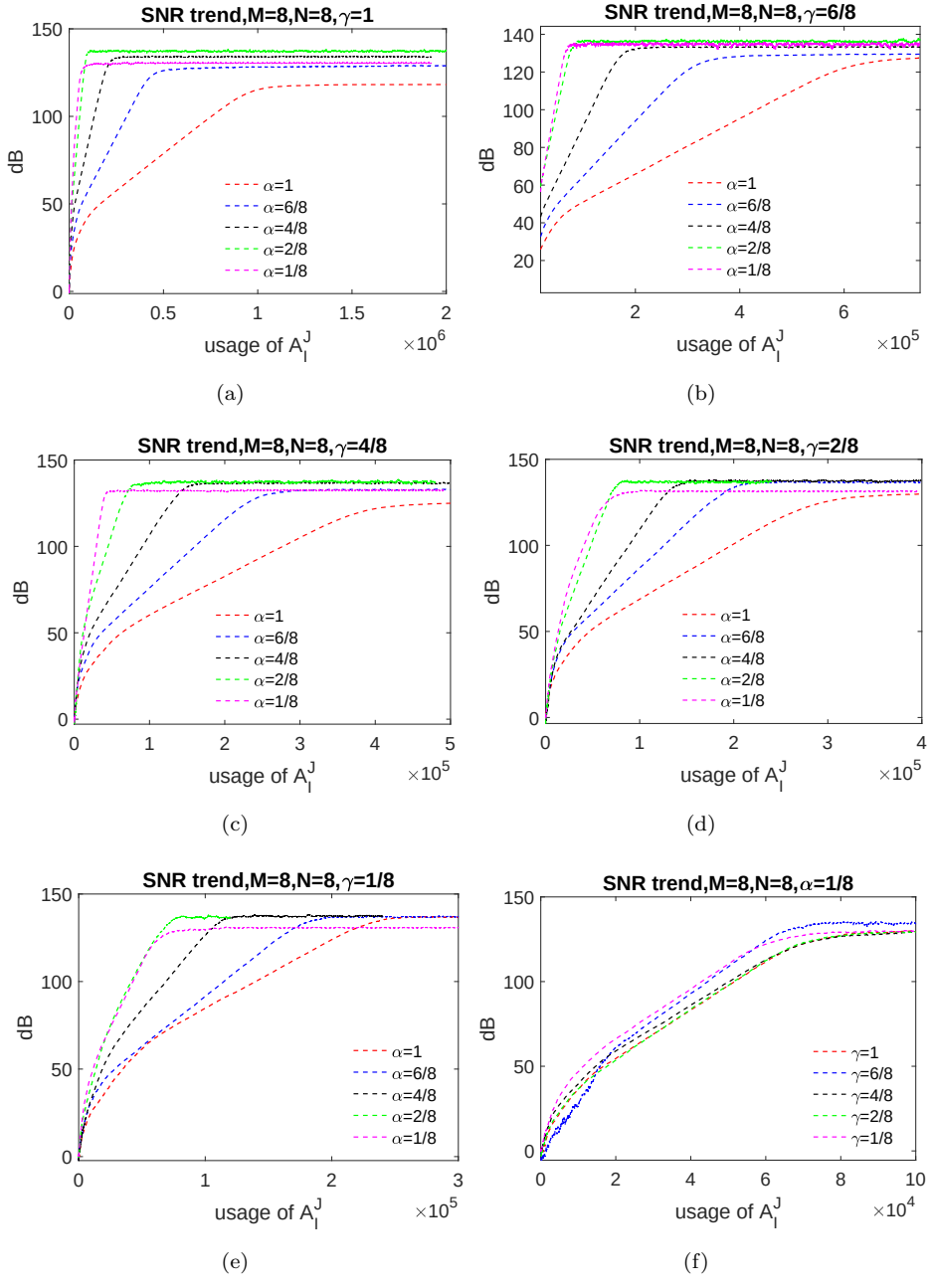


Figure 3.12: For different γ, reducing α contributes to a faster reconstruction speed. (f) illustrates that when $\alpha = \frac{1}{M}$, different γ does not cause significant difference in terms of reconstruction speed.

much and reducing α helps to increase the reconstruction speed at different γ settings.

This is mainly because that the smaller α leads to a more frequent update on the image space.

The above results can be repeated for larger problems: K is 64, OP and OD is 300. Angle gap is 2° and detector pixel number is 400 with pixel length being 0.5. $\mathbf{A} \in \mathbb{R}^{72000\times4096}$. The simulation results are indicated in Fig.3.13. Besides, it is worthy mentioning that this simulation dataset will be repeatedly used in the following demonstration. Results in Fig.3.10, Fig.3.11, Fig.3.12 and Fig.3.13 all show that $\alpha = \frac{1}{M}$ is the best choice among all γ setting and γ does not significantly influences the reconstruction speed when the α is determined. As a result, the suggested α, γ setting method is:

$$\begin{aligned} \gamma &= \min\{1, \tfrac{p}{N}\}, \\ \alpha &= \tfrac{p}{MN\gamma}, \end{aligned} \tag{3.8}$$

where p is the number of parallel nodes in a network.

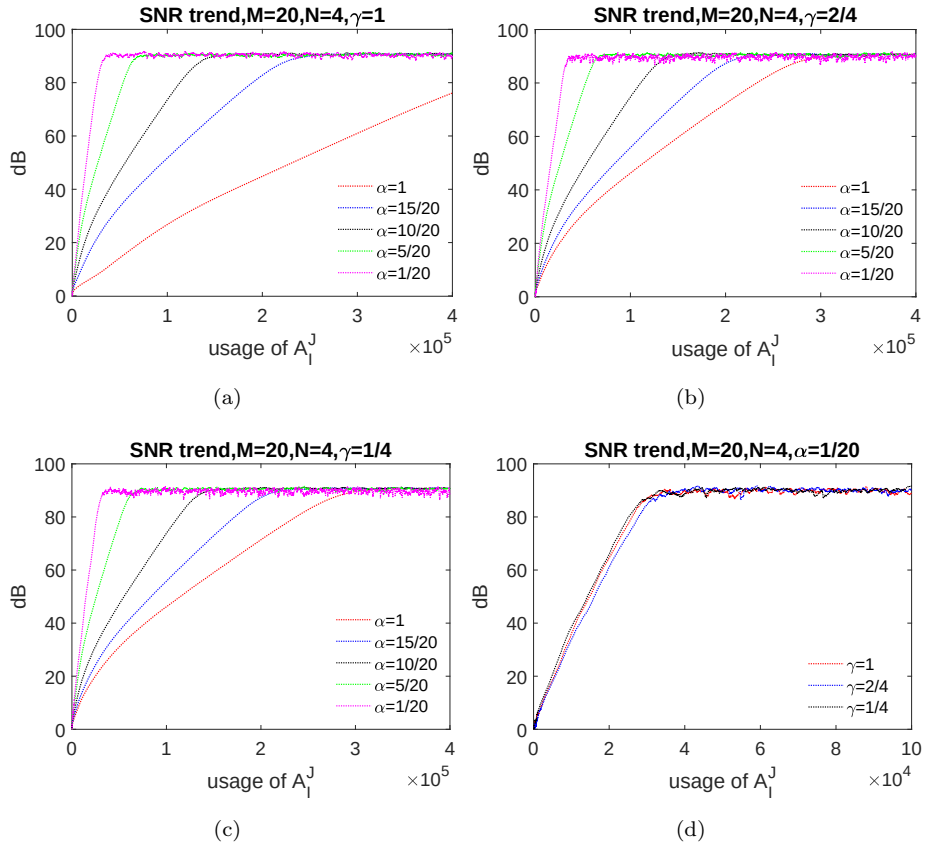


Figure 3.13: All simulations are performed within the β range [0.1,1]. The shape of $\mathbf{A}_I^J$ is TT case. $\alpha = \frac{1}{M}$ is still the best choice among all γ setting in the larger CT model. (d) indicates that when α is set as $\frac{1}{M}$, reducing γ does not significantly influence the reconstruction speed. This property makes the setting of γ flexible. If number of parallel node p is smaller than N, setting $\gamma = \frac{p}{N}$ is thus recommended.

In this section, the reconstruction speed of CSGD under different partitions and different α, γ settings has been discussed. Simulations indicate that CSGD generates iterations that approach the true image in the noise free case within a wide range of M, N settings. Despite the M, N settings influencing the shape of the sub-matrix $\mathbf{A}_I^J$, they do not significantly influence the reconstruction speed. As a result, when partitioning the projection data and image into M and N blocks respectively, the only factor that needs to be considered is whether the partitioned data block fits the storage space of the computation nodes. For different γ settings, $\alpha = \frac{1}{M}$ always leads to the fastest reconstruction speed.

3.2.2 Comparison of CSGD and block ADMM

As discussed in section 2.4, the block ADMM method shares the same partition method on the system matrix $\mathbf{A}$ as CSGD. To facilitate the following comparison of CSGD and block ADMM, a random tiny system matrix $\mathbf{A} \in \mathbb{R}^{256\times128}$ and a random vector $\mathbf{x} \in \mathbb{R}^{128\times1}$, whose elements are both uniformly distributed random numbers in the interval (0,1), are used to reduce the computation amount in simulations. When using large CT models, all simulation results presented in the following are still reproducible.

3.2.2.1 Computation efficiency comparison

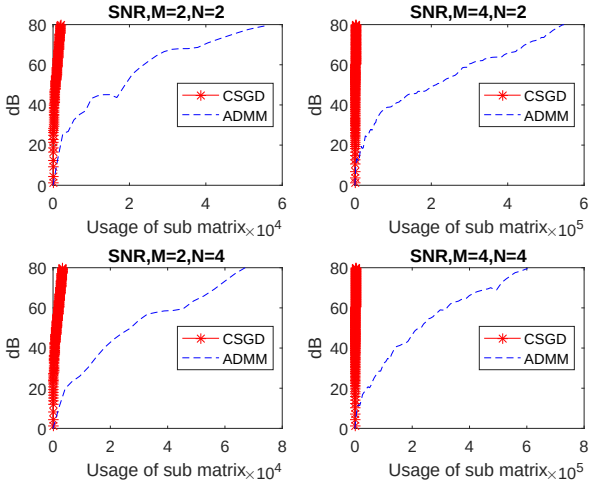
Both CSGD and Block ADMM use consecutive columns and rows in the partitions. They are stopped when their solutions are of 80 dB SNR. For block ADMM, we used two loops to determine the best ρ and τ values that led to the fastest reconstruction speed. The outside loop is for different initial ρ which is between 0.1 and 20 and the inside loop is for τ which is between 0.1 and 0.9. For different partitions, the best parameters leading to the least matrix-vector multiplies are indicated in Table 3.2.

Table 3.2: The best parameter in block ADMM.

	ρ	τ
M=2,N=2	7	0.3
M=2,N=4	2	0.2
M=4,N=2	8	0.9
M=4,N=4	12	0.7

The comparisons of CSGD and block ADMM are indicated in Fig.3.14. It can be seen that CSGD uses much less matrix-vector multiplication. This is because that the block ADMM requires a graph projection procedure, as shown in Eq.2.54, which requires several extra matrix-vector-involved iterations to solve. The CSGD thus become more efficient and use less usage of $\mathbf{A}_I^J$ than the block ADMM. In the figure, the $\alpha = \gamma = 1$.

It should be noted that the fully distributed form is not the instance when the CSGD is the most efficient. According to the previous simulations, $\alpha < 1$ situation can further increase the computation efficiency in CSGD, and then the CSGD should outperform the block ADMM more significantly.



(a)

Figure 3.14: The y-axil is the SNR of iterated $\mathbf{x}$. The x-axil is the usage of $\mathbf{A}_I^J$ and the basic unit is 1×10^4 times. The $\alpha = \gamma = 1$. The CSGD algorithm greatly reduces the required matrix vector operations to meet the predefined SNR than the ADMM method.

3.2.2.2 Communication cost and storage demand comparison

The above simulation indicates that to obtain a solution with a predefined SNR, CSGD requires much fewer matrix-vector multiplications, which means fewer computations in each computation node. This is an advantage of CSGD over the block ADMM method. Another aspect to be compared is the communication cost. To demonstrate the difference between block ADMM and CSGD, let M and $N = 2$. An illustration of the work procedure of block ADMM is indicated in Fig.3.15. Each computation node (ovals in

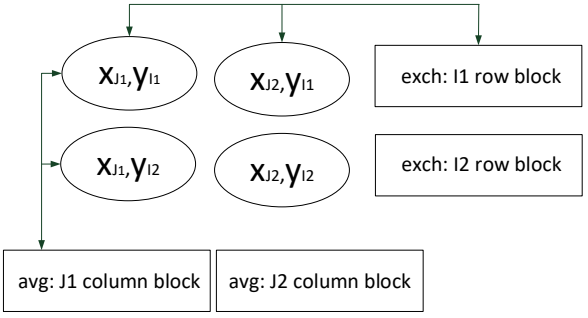


Figure 3.15: The work scheme of block ADMM method

the figure) keeps partial $\mathbf{x}_J$ and $\mathbf{y}_I$. There are two types of master node (square box in the figure). One type performs "avg" procedure for column blocks and the other type performs "exch" procedure for row blocks. (Their definitions and functions have been explained in section 2.4) For the left-bottom master node, it stores $\mathbf{x}_{J_1}^{k+\frac{1}{2}}$, $\mathbf{x}_{J_1}^k$ and $\tilde{\mathbf{x}}_{J_1}^k$. For the right-top master node, it stores $\mathbf{z}_{I_1}^{k+\frac{1}{2}}$, $\mathbf{z}_{I_1}^k$ and $\tilde{\mathbf{z}}_{I_1}^k$. The work scheme can be divided into five parts: The first part happens in four master nodes: bottom two master nodes perform $prox_{\lambda g_j}(\mathbf{x}_{J_j}^k - \tilde{\mathbf{x}}_{J_j}^k) = \mathbf{x}_{J_j}^k - \tilde{\mathbf{x}}_{J_j}^k$ (defined in Eq.2.55) and right two master nodes perform $prox_{\rho f_i}(\mathbf{z}_{I_i}^k - \tilde{\mathbf{z}}_{I_i}^k)$ (defined in Eq.2.56). The second part happens in four computation nodes for Π projection. These two parts can be performed in parallel. After these two parts, the "avg" and "exch" parts happens in four master nodes again. For the left-bottom master node, it communicates with the corresponding computation node to ask for ${\mathbf{x}_{J_1}^i}^{k+\frac{1}{2}} (i \in [1,2])$ which exist in each computation node. After the average process the master returns $\mathbf{x}_{J_1}^{k+1}$ to the computation nodes. Similarly, the right-top master node communicates with the corresponding computation nodes to ask for ${\mathbf{z}_{I_1}^j}^{k+\frac{1}{2}} (j \in [1,2])$. After the "exch" calculation, the calculated ${\mathbf{z}_{I_1}^j}^{k+1} (j \in [1,2])$ is reassigned to the corresponding computation nodes. The following dual update on $\tilde{\mathbf{x}}_{J_j}^{k+1}$, $\tilde{\mathbf{z}}_{I_i}^{k+1}$ happens in the master nodes and the update on $(\tilde{\mathbf{x}}_{J_j}^i)^{k+1}$ happens in the computation nodes.

The same distributed architecture is also suitable for the CSGD method, although in the previous CSGD algorithm description, a so called "high level master node" is introduced. In fact, the "high level master node" is not necessary in reality and after removing the "high level master node", the work scheme of CSGD is indicated in Fig.3.16.

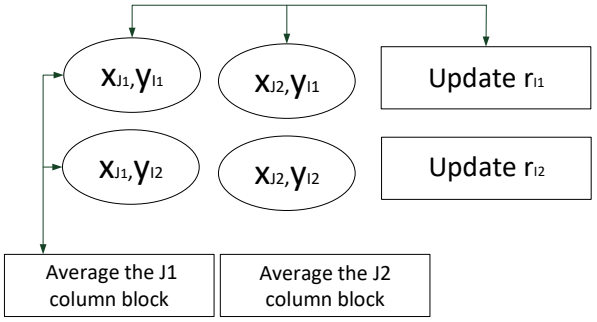


Figure 3.16: The work scheme of CSGD method, which can be the same as for the block ADMM.

For the bottom master nodes, they do not have to store any variables and they simply communicate with the corresponding computation nodes and average column blocks. For the right side master nodes, they only store the corresponding projection data $\mathbf{y}_I$. For example, the right-top master node only stores $\mathbf{y}_{I_1}$. To update $\mathbf{r}_{I_1}$, the master node asks for $\mathbf{z}_{I_1}^j (j \in [1,2])$ from corresponding computation nodes. It is interesting to find that although $\mathbf{z}_I^j$ has different meanings in CSGD and block ADMM, the size of both

is equal to the block dataset $\mathbf{y}_I$. As a result, the communication cost of CSGD and ADMM is the same.

The storage demand for the master node, as discussed above, is different between the two nodes. Further analysis indicates that for the computation nodes, the storage demand of CSGD is also lower than for block ADMM. Table 3.3 clearly indicates the comparison results.

Table 3.3: Storage demand for each node in CSGD and block ADMM

	CSGD	ADMM
left-bottom(master node)	None	$\mathbf{x}_{J_1}^k$,$\tilde{\mathbf{x}}_{J_1}^k$ and $\mathbf{x}_{J_1}^{k+\frac{1}{2}}$
right-top(master node)	$\mathbf{y}_{I_1}$	$\mathbf{z}_{I_1}^k$,$\tilde{\mathbf{z}}_{I_1}^k$ and $\mathbf{z}_{I_1}^{k+\frac{1}{2}}$,$\mathbf{y}_{I_1}$
left-top(computation node)	$\mathbf{x}_{J_1}^k$,$\mathbf{y}_{I_1}$,$\mathbf{z}_{I_1}^1$	${\mathbf{x}_{J_1}^1}^{k+\frac{1}{2}}$,${\mathbf{z}_{I_1}^1}^{k+\frac{1}{2}}$,$\mathbf{x}_{J_1}^k$,$(\tilde{\mathbf{x}}_{J_j}^i)^k$,${\mathbf{z}_{I_1}^1}^k$,$\tilde{\mathbf{z}}_{I_1}^k$

3.2.3 Importance sampling approach

In the previous simulations, the partition method adopts a partitioning where blocks use consecutive rows and columns. In a circular scanning geometry, the sequential projection views often provide unnecessarily similar projection information, especially in the case where the projection view gap is small. One way to increase the reconstruction speed is to simply shuffle the projection views and then divide them into several row blocks. This is called ordered subsets method (Hudson and Larkin, 1994). To demonstrate the effectiveness of ordered subsets, the simulation results using $\mathbf{A} \in \mathbb{R}^{72000\times4096}$ are indicated in Fig.3.17 and Fig.3.18, which illustrates $\alpha \geqslant \frac{1}{2}$ and $\alpha < \frac{1}{2}$ situations respectively.

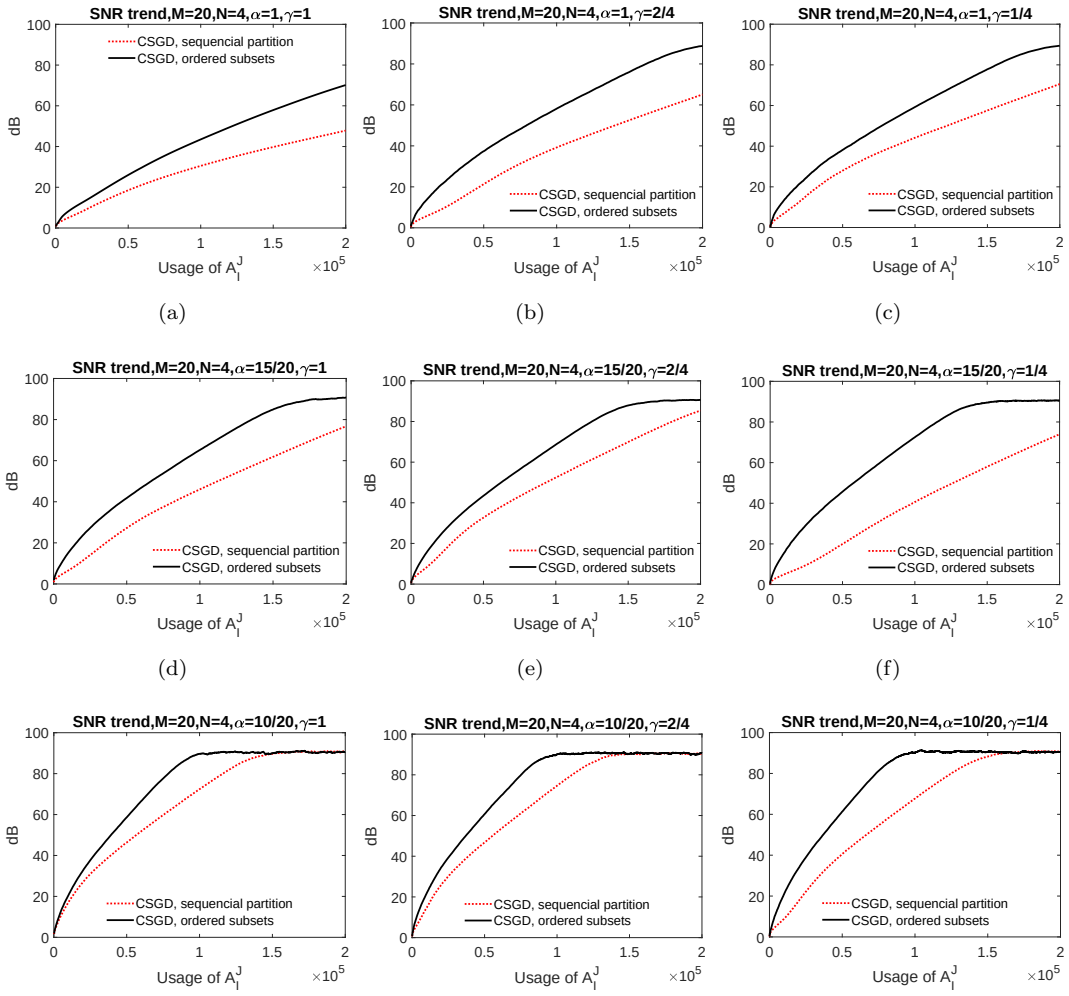


Figure 3.17: When α is $1, \frac{15}{20}, \frac{10}{20}$, the ordered subsets method increases the reconstruction speed.

It can be seen that when α approximately equals $\frac{1}{M}$, the ordered subsets barely increases the reconstruction speed. To further increase the reconstruction speed, an importance sampling method is proposed. This strategy is proposed based on the scanning geometry and the system matrix $\mathbf{A}$'s sparsity property. If the reconstructed volume is properly divided, projections of each sub-volume are then constrained within a relatively small area on the detector. For example, in the 2D image case, if the image is divided into 2×2 sub-images by halving each dimension of the image, projections within all projection views of each sub-image are indicated in Fig.3.19.

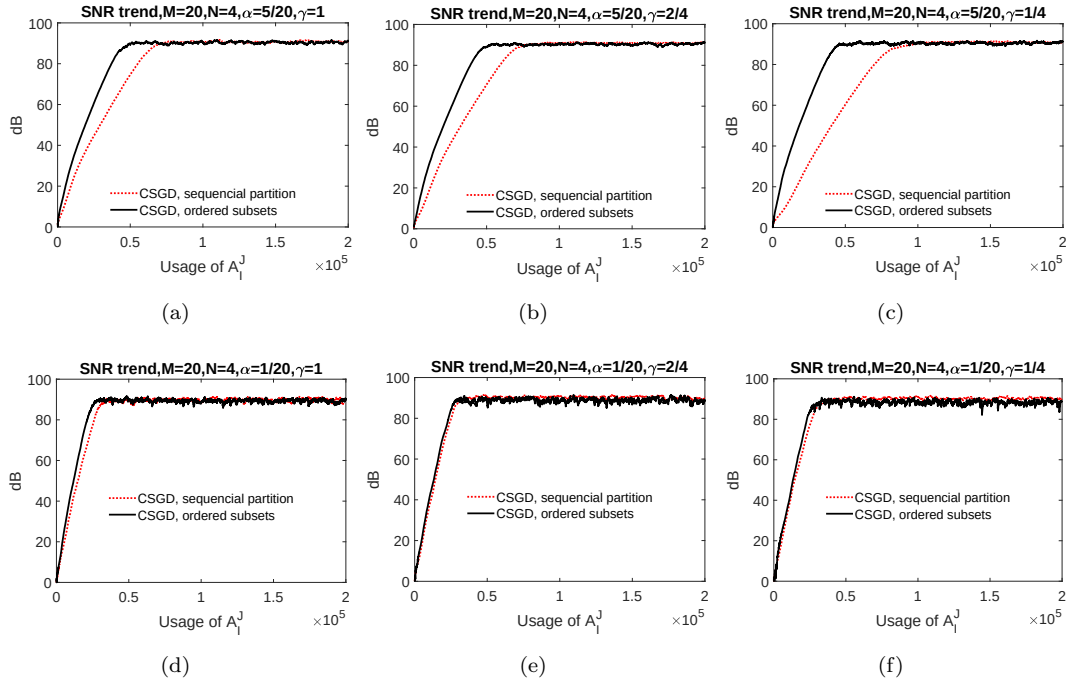
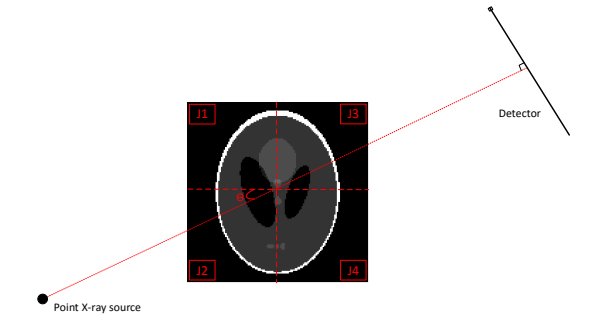


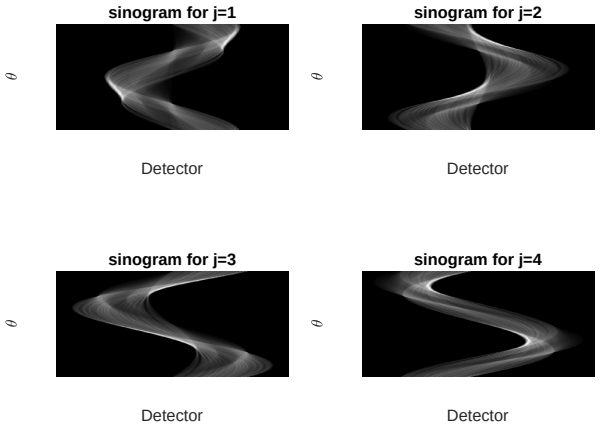
Figure 3.18: When α is $\frac{1}{20}$, the ordered subsets method barely helps to increase the reconstruction speed.

CSGD-IM breaks each single projection into several sub-projections and only samples partial sub-projections from each single projection for the selected sub-volume $\mathbf{x}_J$ (e.g. divide the projection into 2 sub-projections in Fig.3.19(a) and sample 1 sub-projection). This sampling is not done uniformly but is based on a selection criteria that uses the relative sparsity of each matrix $\{\mathbf{A}_{\hat{I}_i}^J\}$, i.e. the denser a sub-matrix is, the higher the probability that it is selected. Here $\hat{I}_i$ means a sub-area of the detector for a certain projection view. As we do not have access to matrix $\mathbf{A}$ in large scale CT scanning scenarios, to estimate the sparsity pattern of $\mathbf{A}$, CSGD-IM computes the fraction of a sub-image's projection area on each sub-detector and these fractions are used as possibility distributions. If a sub-detector has a higher fraction than the other, then it has higher possibility to be selected in iterations. When the row block I_i contains several projection angles, the importance sampling strategy is repeatedly applied to each projection angle contained in the row block. The CSGD-IM algorithm is indicated in Algo.3.4.

The advantage of CSGD-IM is that it provides each computation node more projection views within one row block than CSGD and thus reduces the row block number M. For example, assume that the detector contains N_d detector pixels and each divided sub-image contains N_p pixels. The size of each element in both $\mathbf{x}$ and $\mathbf{y}$ space is b bytes. The computation node's storage capacity is assumed to be $b \times (N_d + N_p)$. This means that each time the original CSGD is only allowed to process and address one projection view for one selected sub-image and thus M should equal the projection view number.



(a)



(b)

Figure 3.19: (a) By slicing each dimension of image into 2 parts, the image is divided into 4 parts.(b) The sinogram indicates that the projection of each sub-image is constrained into a small area (approximately half) of the detector for all projection views θ.

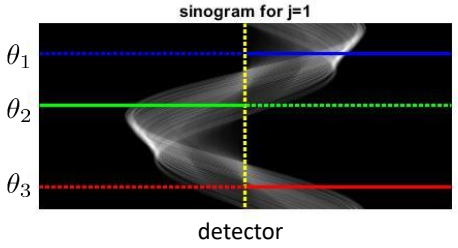
CSGD-IM with the partition of Fig.3.19 can use two projection angles by choosing one detector sub-areas for each projection angle, thus halving M. According to the previous simulations indicated in Fig.3.8, a smaller M can increase the reconstruction speed especially when M is already very large compared with N.

However, simply dividing the detector into 2 sub-areas and sampling 1 sub-area makes some sub-detectors hard to be selected because of the relatively small projection fraction, as illustrated in Fig.3.20(a). It implies that in the following iterations, when it comes to $\mathbf{x}_{J_1}$ and $\mathbf{y}_{I_1}$, the sub-detectors reflected by solid lines will be more frequently selected than dashed lines. This fact means $\mathbf{z}^1_{I_d}$ cannot be updated fully and timely, where I_d here is a subset of I_1 for those dashed lines. According to line 14 in Algo.3.3, $\mathbf{z}^1_{I_d}$ is used to update $\mathbf{r}_{I_d}$ together with $\mathbf{z}^{j=2,3,4}_{I_d}$. At the initial iteration stage, the inaccurate and outdated $\mathbf{z}^1_{I_d}$ does not influence much for roughly reducing $\|\mathbf{r}_{I_d}\|$ since the other three $\mathbf{x}_{J_j=2,3,4}$ will generate most of the projection information. However, when the iteration goes on and $\|\mathbf{r}_{I_d}\|$ has been reduced to nearly zero (in the noise free model), then an accurate and updated $\mathbf{z}^1_{I_d}$ is required, although $\mathbf{z}^1_{I_d}$ contains loads of zero projection

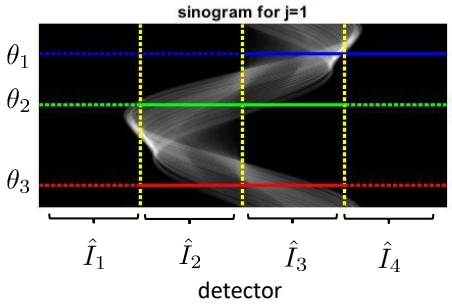
Algorithm 3.4 CSGD with importance sampling strategy.

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ is the percentage of the selected row and column blocks respectively.
for *epoch* $= 1, 2, ..., K_{max}$ **do**
  $\hat{\mathbf{x}} = \mathbf{0}$
  for jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
    Random select a column block from sets $\{J_j\}_{j\in[1,N]}$ with replacement
    for ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
      Random select a row block from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      Importance sample one sub-detector area from each single projection view. All indexes represented by those selected sub-detector areas form the row index set $\hat{I}_t$.
      $\mathbf{g}_J = (\mathbf{A}^J_{\hat{I}_t})^T \mathbf{r}_{\hat{I}_t}$
      $\mu = \beta \frac{(\mathbf{g}_J)^T \mathbf{g}_J}{(\mathbf{g}_J)^T (\mathbf{A}^J_{\hat{I}_t})^T \mathbf{A}^J_{\hat{I}_t} \mathbf{g}_J}$
      $\mathbf{z}^j_{\hat{I}_t} = \mathbf{A}^J_{\hat{I}_t} (\mathbf{x}_J + \mu \mathbf{g}_J)$
      $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu \mathbf{g}_J$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N} \mathbf{z}^j$
  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
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

data. To avoid some sub-detectors being rarely selected for specific sub-images, in the 2D fan-beam case, the detector can be divided into 4 sub-detectors and each time we select 2 adjacent sub-detectors, as indicated in Fig.3.20(b).



(a)



(b)

Figure 3.20: Slicing detector into 2 or 4 sub-areas in importance sampling. (a) Assume that one computation node requests $\mathbf{x}_{J_1}$ and $\mathbf{y}_{I_1}$ containing 3 projection views θ_1, θ_2 and θ_3. Instead of receiving all 3 full projections, the computation node samples 3 sub-detectors from each view. Here the solid lines stand for sub-detector areas that have high probability to be selected because most of the sinogram information falls on them. Dashed lines stand for the sub-detector areas that have low probability to be selected due to the small sinogram fraction. (b) $\hat{I}_{1-4}$ present 4 different sub-detectors. In the previous importance sampling method, only $\hat{I}_1 + \hat{I}_2$ or $\hat{I}_3 + \hat{I}_4$ can be selected. Now, a more flexible sampling method selects 2 adjacent sub-detectors each time. For θ_2 view, the possibility to choose $\hat{I}_1 + \hat{I}_2$ is almost the same as $\hat{I}_2 + \hat{I}_3$, thus avoid making $\mathbf{z}^1_{\hat{I}_3,\theta_2}$ overly outdated.

The following simulations indicate that the importance sampling by dividing the detector into 2 or 4 sub-detectors can increase the reconstruction speed, as indicated in Fig.3.21 and Fig.3.22. The simulation data still adopts $\mathbf{A} \in \mathbb{R}^{72000\times 4096}$ in this case. Simulations have verified that the importance sampling method which divides the detector into 4 sub-areas and sample 2 adjacent sub-detectors outperforms the ordered subset CSGD for all α, γ settings. The importance sampling represented by blue lines, however, only accelerates the reconstruction speed at the initial stage. When the SNR is higher than 20 dB, then it stops to outperform the original CSGD because of those rarely updated $\mathbf{z}^j_{I_d}$.

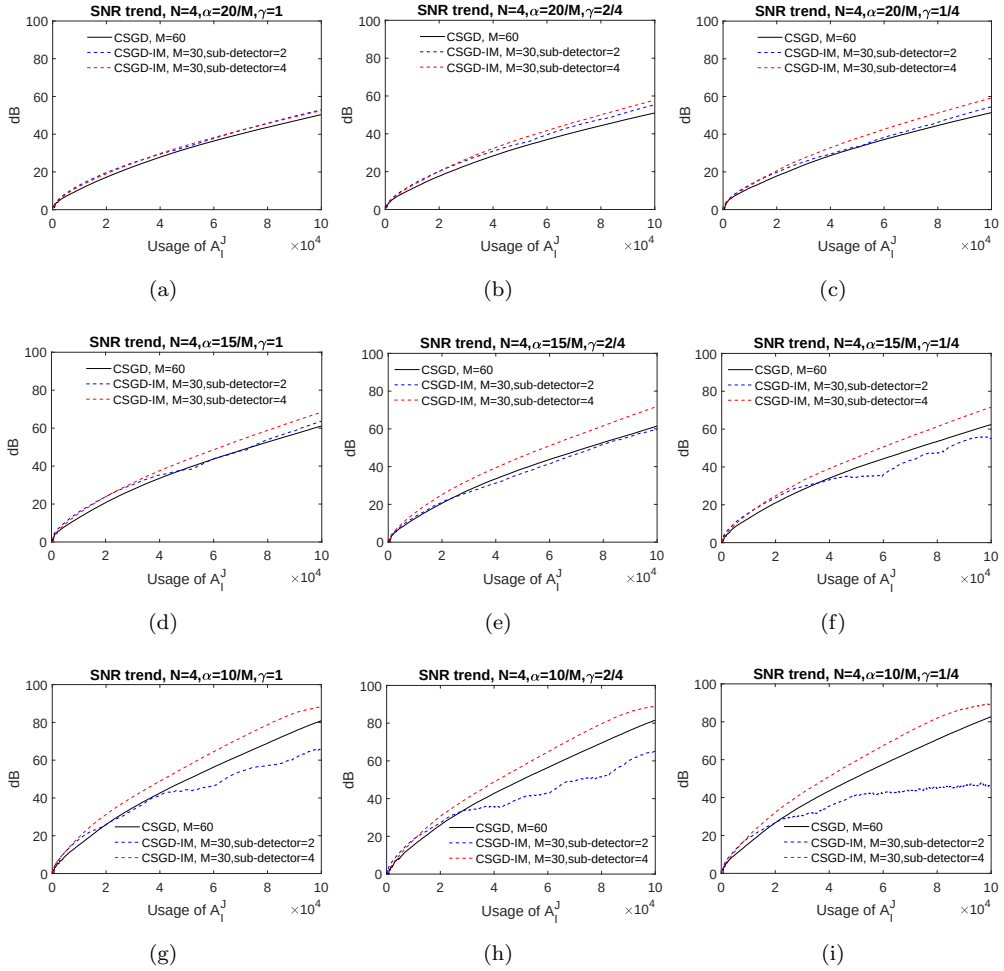


Figure 3.21: Three methods' reconstruction speed is compared. Black lines are CSGD with ordered subsets, which include shuffled projection views with one row block. Red lines represent the importance sampling method which divides the detector into 4 sub-areas and samples 2 adjacent sub-detectors. Blue lines are the importance sampling method which divide the detector into 2 sub-areas and samples 1 sub-area each time. Under all α, γ settings the importance sampling which divide the detector into 4 sub-areas outperform the other two reference methods.

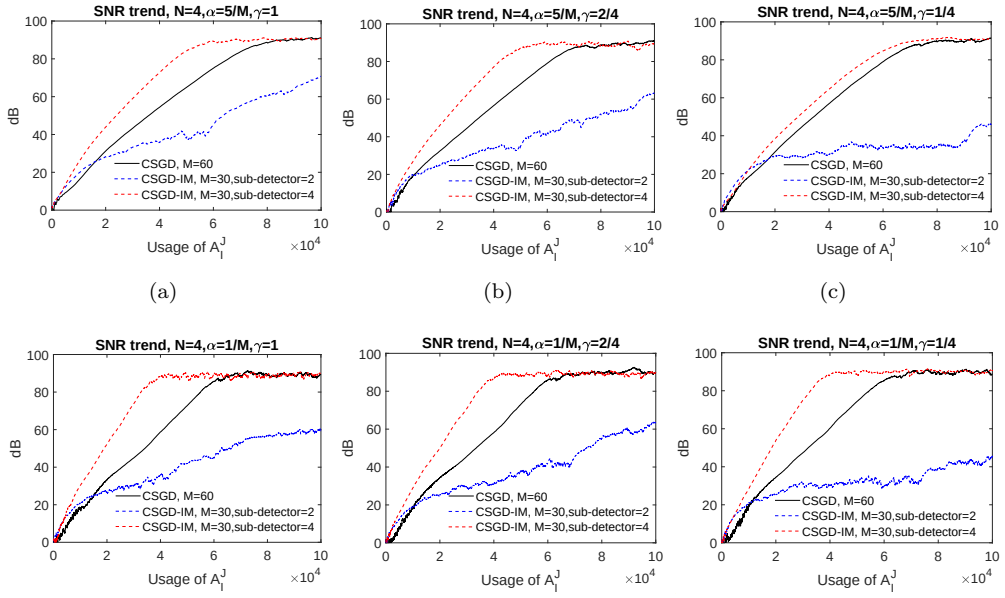


Figure 3.22: When α is reduced, the trend reflected in Fig.3.21 still holds. The importance sampling which divide the detector into 4 sub-detectors and sample 2 adjacent parts is of the fastest reconstruction speed among the others.

The importance sampling method also works for 3D cone beam data. Slicing of the volume and of the corresponding projection area is indicated in Fig.3.23. Another slicing

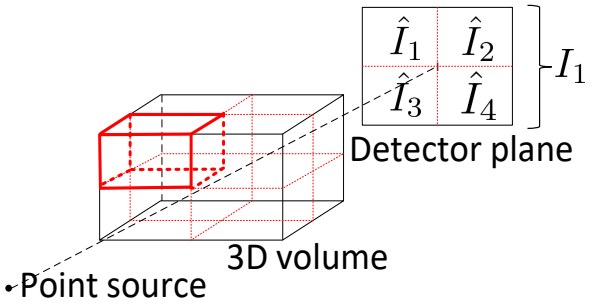


Figure 3.23: Example cone beam CT setting for the first projection view, where the 3D volume rotates horizontally. The 3D volume is divided into 8 sub-volumes. The projection of one sub-block is mainly concentrated in a small area on the detector. If the detector is also divided into 2×2 sub-areas, then the currently selected sub-volume (bold red frame) is mainly projected onto the top-left and top-right sub-detector area, i.e. $\hat{I}_1$ and $\hat{I}_2$ area.

methods on the detector is to slice it into 2×4 detectors and each time selecting 2 adjacent sub-detectors, as illustrated in Fig.3.24.

Importance sampling in 3D scanning case only samples a quarter of a full projection, thus the maximum allowed projection views is 4 times larger than with ordered subsets. To verify the property of importance sampling, a cone-beam circular scanning model

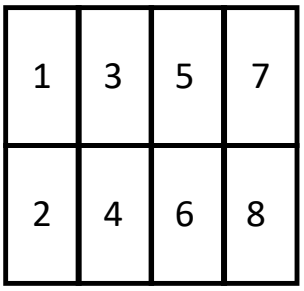


Figure 3.24: The detector is divided into 2×4 sub-detectors and when sampling the sub-detector, there are 6 areas in total to be considered as candidates: (1,3),(3,5),(5,7),(2,4),(4,6),(6,8)

shown in Fig.1.2(c) is adopted and the source-detector starts with scanning from the left of the volume and then rotates around the object clockwise. The parameter is set as $OP = 1000, OD = 536$; 3D volume is a $16 \times 16 \times 16mm^3$ cubic containing 64^3 voxels; The detector is a $50 \times 50mm^2$ square panel containing 400^2 detector pixels. The scanning interval is 1°. The simulations are indicated in Fig.3.25. To shorten the simulation time, all simulations are terminated before the optimal result was reached. The first row of Fig.3.25 is the reconstruction speed of CSGD under different α, γ settings, indicating that for each γ case, $\alpha = \frac{1}{M}$ is the fastest form. When comparing CSGD and CSGD-IM, only $\alpha = \frac{1}{M}$ is presented in the second row. It indicates that the importance sampling with 2×4 slicing detector method is the fastest method under different γ settings.

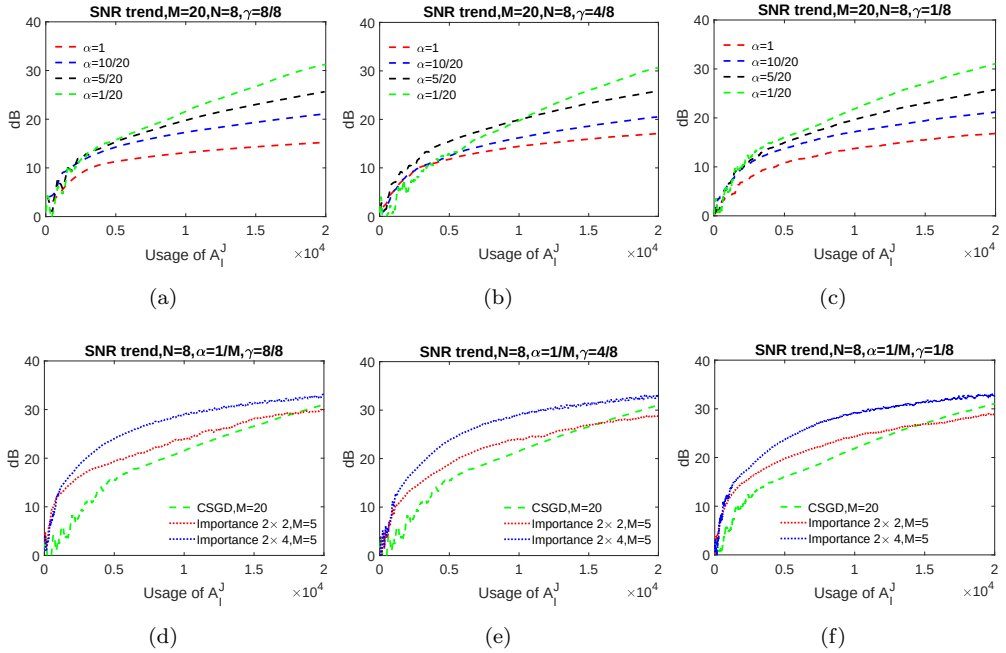


Figure 3.25: The y-axis is dB and x-axis is "usage of $\mathbf{A}_I^J$". The first row reflect the reconstruction speed of CSGD under different α, γ settings. The second row reflects the comparisons of CSGD with ordered subsets and importance sampling methods with two different partitions on the detector.

3.3 Computational complexity

There are several important aspects when comparing computational efficiency of the methods. The methods are designed to allow parallel computation. It is envisaged that this will be performed on a distributed network of computation nodes. Computation nodes produce two outputs, as displayed below.

1. $\hat{\mathbf{x}}_J = \mathbf{x}_J^k + \mu \mathbf{g}_J$
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. $\mathbf{x}_J = \text{mean}_i(\hat{\mathbf{x}}_J)$
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.

Three aspects affect performance:

1. Computational complexity in terms of multiply-adds.
2. Data transfer requirements between data storage and processing units as well as between different processing units.
3. Data storage requirements, both in terms of fast access random access memory (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 $\mathbf{A}_I^J$ 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 $O(|I| \times |J|)$, 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 $\mathbf{x}$. It is also needed to compute and store averages/sums over $\hat{\mathbf{x}}_J^i/\mathbf{z}_I^j$. These computations can be performed efficiently using parallel data reduction techniques. The proposed approach would mean that each node would thus require $O(|I|+|J|)$ local memory.

3.4 Fixed step CSGD

At the first glance on CSGD, it seems that the step-length $\mu = \beta\frac{(\mathbf{g}_J)^T\mathbf{g}_J}{(\mathbf{g}_J)^T(\mathbf{A}_I^J)^T\mathbf{A}_I^J\mathbf{g}_J}$ can be further simplified and make it a constant $\mu = \beta$. This simplification makes the algorithm more concise and the mathematical analysis simpler. The algorithm is indicated in Algo.3.5. Due to the fact that μ is a constant, the pseudocode in line 10 can be moved

Algorithm 3.5 CSGD algorithm which uses a constant step-length.

1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ is the percentage of the selected row and column blocks respectively, μ is a constant.
2: **for** *epoch* $= 1, 2, ..., K_{max}$ **do**
3: $\hat{\mathbf{x}} = \mathbf{0}$
4: **for** jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
5: Random select a column block J from sets $\{J_j\}_{j\in[1,N]}$ with replacement
6: **for** ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
7: Random select a row block I from sets $\{I_i\}_{i\in[1,M]}$ with replacement
8: $\mathbf{g}_J = (\mathbf{A}_I^J)^T\mathbf{r}_I$
9: $\mathbf{z}_I^j = \mathbf{A}_I^J(\mathbf{x}_J + \mu\mathbf{g}_J)$
10: $\hat{\mathbf{x}}_J = \hat{\mathbf{x}}_J + \mathbf{x}_J + \mu\mathbf{g}_J$
11: **end for**
12: **end for**
13: $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N}\mathbf{z}^j$
14: For all blocks J that have been updated, $\mathbf{x}_J = \hat{\mathbf{x}}_J/\alpha M$
15: **end for**
16: $\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

out of the for-loop and then the algorithm is indicated in Algo.3.6. Actually, Algo.3.6 is the first algorithm proposed during this project. However, this simpler version is of a slow reconstruction speed, despite the number of matrix-vector multiplications per epoch decreases from $3\alpha\gamma MN$ to $2\alpha\gamma MN$. For example, in a tiny CT scanning system whose system matrix is indicated as Fig.3.6, the reconstruction speed of CSGD with constant step-length and the range of step-lengths is shown in Fig.3.26. It shows that using a constant step-length in CSGD impedes the original CSGD's performance. The range of step-length leading to high accuracy solution also becomes narrower than before, making μ harder to be tuned. The simulation result justifies the importance of calculating the step length μ using Eq.3.7. Despite the slow reconstruction speed, this simpler fixed

Algorithm 3.6 CSGD algorithm which uses constant step-length.

Initialization: $\{\hat{\mathbf{g}}^i\}_{i\in[1,\alpha M]} = \mathbf{0}$. $\mathbf{g} = \mathbf{0}$. The other initialization is the same with Algo.3.5.
for *epoch* $= 1, 2, ..., K_{max}$ **do**
  $\{\hat{\mathbf{g}}^i\}_{i\in[1,\alpha M]} = \mathbf{0}$
  for jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
    Random select a column block J from sets $\{J_j\}_{j\in[1,N]}$ with replacement
    for ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
      Random select a row block I from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      $\hat{\mathbf{g}}_J^{ii} = (\mathbf{A}_I^J)^T \mathbf{r}_I$
      $\mathbf{z}_I^j = \mathbf{A}_I^J(\mathbf{x}_J + \mu\hat{\mathbf{g}}_J^{ii})$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N} \mathbf{z}^j$
  For all blocks J that have been updated,$\mathbf{g}_J = \sum_{ii=1}^{\alpha M} \hat{\mathbf{g}}_J^{ii}$
  For all blocks J that have been updated, $\mathbf{x}_J = \mathbf{x}_J + \frac{\mu}{\alpha M}\mathbf{g}_J$
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

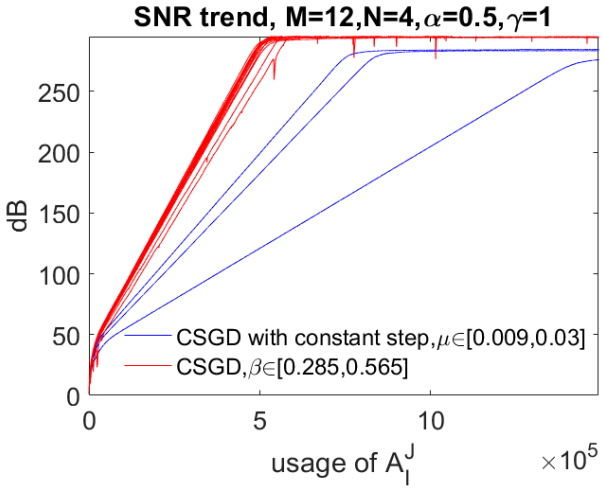


Figure 3.26: A tiny, noise free CT scanning geometry indicated as Fig.3.6 is used here. Red lines are repeated reconstruction speeds when CSGD using a constant β (i.e. calculate μ using Eq.3.7) and blue lines are speeds when CSGD using a constant μ.

step CSGD which accumulates partial stochastic gradient ingredients is illuminating for the next proposed algorithm, which is empirically shown to approach the true solution in the noise free setting even with a constant step-length.

3.5 Conclusions

This chapter presents the first parallel algorithm CSGD. It is 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.

The properties of CSGD have been studied in details. Simulations indicate that when the 2D or 3D object is scanned by a circle trajectory and with sufficient projection views, the shape of sub-matrix $\mathbf{A}_I^J$ (TT,SF or square) does not significantly influence the reconstruction speed. As a result, the division rule on the dataset and volume is not rigid and the only requirement for dataset division is that the divided projection data and volume can be small enough to be stored by those separate computation nodes. The CSGD allows arbitrary α, γ selection criteria and they all lead to converging results. The reconstruction speeds differ for different α, γ settings. Simulations experimentally indicate that for different γ settings, $\alpha = \frac{1}{M}$ leads to the fastest reconstruction speed. An importance sampling strategy have been developed, which has been shown to further increase the reconstruction speed. In the original CSGD, since the image is divided into N sub-images, projections of them are not distributed on the whole detector but on a limited sub-detector area. If the scanning geometry for each sub-volume still adopts the whole geometry situation, a lot of zero projection results will be generated. These zero projections have little information and often contribute little to the reconstruction. The importance sampling method divides the detector into many sub-detectors and each time only samples partial of sub-detectors. This enables each row block to contain more projection views and let sub-detectors with massive non-zero projection data to be used more frequently in iterations than those with loads of zero projection data.

The parallel architecture can be 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 reconstructs the image to a predefined precision with significantly fewer matrix vector products as it avoids the calculation of matrix inverses. This means that the CSGD has much higher computation efficiency than block ADMM.

One open area of this chapter is the formal analysis of the convergence property CSGD. This chapter mainly focuses on the properties of the CSGD under different parameter settings and prove that the CSGD is able to present an increment SNR trend of the reconstructed image to a high accuracy solution, showing its ability to reduce the data fidelity. In the noise free model, CSGD with any α, γ selection criteria can all quickly achieve a 80dB and even higher reconstruction results, which is already enough to present inner details of the reconstructed image. When the Gaussian-type noise is added, the CSGD algorithm is also able to generate an approximately stable result. However, where

the results is is not fully understood yet. A conjecture is that the CSGD should converge to the least square solution with an error bound. Some efforts have been made to explore the convergence property of CSGD, which will be presented in Appendix A.

In this thesis, the simulation results for projection data corrupted by severe noise are not presented. Those simulations show significant semi-convergence phenomenon. With a proper early termination strategy, for example, terminating the iteration when the SNR has achieved its peak value, the CSGD shows similar relative reconstruction speed comparison results under different α, γ parameter settings shown in Fig.3.9 to 3.12, and the simulations also prove the claim made in Eq.3.8 for how to determine α, γ for a fast reconstruction speed. However, after introducing the early stopping strategy, it then becomes difficult to convince readers that the CSGD are converging. Furthermore, in simulations, when to determine the CSGD is easy to determine, as the SNR trend is visible (the $\mathbf{x}_{true}$ is a known vector), but in realistic when to determine the iteration is difficult and it becomes another research topic. Consider the fact that the main focuses of Chapter 3 are to illustrate that the CSGD has the ability to reduce the data fidelity and obtain an increment SNR trend and to show the recommended parameter(mainly α, γ) setting, this thesis does not present the severe noise simulations whose semi-convergence is significant. The detailed discussions on semi-convergence property of CSGD thus is another open area of this thesis.

Chapter 4

BSGD

4.1 Block Stochastic Gradient Descent

In this chapter, another parallel algorithm is proposed. This algorithm is inspired by SAG, which stores and sums up all previously calculated sub-gradients. The new algorithm can be viewed as a combination of CSGD and SAG and is called the "Block Stochastic Gradient Descent" (BSGD) method. It belongs a SGD type algorithm but the computation overhead per iteration is further reduced.

The contributions of this chapter includes: 1) This algorithm is also specially designed for the case where both projection data and reconstruction volume are so large that one computation node only has partial access to both. However, different from CSGD, the new proposed BSGD sums up all previous calculated sub-gradients. Simulations show that the calculated gradient has a decreasing error variance compared with the true gradient direction (i.e. the expectation of $\|\mathbf{g}^k - \mathbf{g}^k_{true}\|$ decreases on expectation with each iteration, where $\mathbf{g}^k, \mathbf{g}^k_{true}$ are calculated gradient and the true gradient at k^{th} iteration). 2) Comparisons of BSGD with CSGD and other SGD-type algorithms is presented, including reconstruction speed, storage capacity and communication schemes in the parallel networks. Simulations further verify that the importance sampling trick proposed in Chapter 3 is also applicable to BSGD, and it shows that this version of BSGD has the fastest reconstruction speed among all reference algorithms. 3) Most of SIRT-type iteration, including BSGD, requires a well tuned step length. If the step length is set improperly small, the BSGD often needs enormous matrix-vector multiplications to reconstruct the image into predefined precision. To overcome this difficulty, an automatic parameter tuning trick is proposed to dynamically enlarge the step-length to a range that makes the update of the image vector $\mathbf{x}_{rec}$ more aggressive than small step length case whilst empirically enabling the iteration approach the least squares solution. Simulation results show that the accelerated algorithm reconstructs the image to the same quality with much less matrix-vector multiplications than original BSGD with constant small

step lengths. 4) The proposed automatic parameter tuning mainly works for the case when initial step-length is improperly small. If the initial step is well tuned, this tuning strategy cannot significantly improve the performance. In this case, a recent acceleration technique KatyushaX^s is first applied on BSGD and is shown to be effective. The KatyushaX^s acceleration is similar to Nesterov acceleration by gradually updating the step length and thus it does not introduce extra expensive computation and it shows that this acceleration method works well with BSGD.

4.2 BSGD algorithm

The new proposed algorithm is named as Block Stochastic Gradient Descent (BSGD). The inspiration of BSGD comes from the fixed step CSGD, which divides $\mathbf{y}, \mathbf{x}$ into M, N blocks respectively and the SAG, which stores the old stochastic gradient ingredients obtained from different row blocks. Similarly, in BSGD, the stochastic gradient ingredients representing the BP results for different $\mathbf{A}_{I_i}^{J_j}$ are also recorded and is named as $\hat{\mathbf{g}}_{J_j}^{i}$. They are summed up as a new update direction. Meantime, FP results for different $\mathbf{A}_{I_i}^{J_j}$ are also recorded into new introduced variables $\mathbf{z}_{I_i}^{j}$, which are used to update the residual $\mathbf{r}$. A difference between BSGD and the other row-action methods or the combinations of row and column-action methods is that BSGD adopts a more stochastic update scheme. Instead of updating $\mathbf{x}$ after and only after the exact residual $\mathbf{r}$ being computed, BSGD computes a stochastic approximation of the residual by only processing a subset of $\mathbf{x}$ in each iteration and using previously computed estimates of $\mathbf{A}_I^J\mathbf{x}_J$ for those blocks that are not used in the current iteration. The hope is that the increase in uncertainty in the gradient estimate is compensated for by a reduction in computation and communication cost. BSGD thus does not compute all $\mathbf{A}_I^T\mathbf{r}_I$ and $\mathbf{A}_I\mathbf{x}_I$ in each iteration. Instead, during each iteration, only αM row indexes from $\{I_i\}_{i=1}^{M}$ and γN column indexes from $\{J_j\}_{j=1}^{N}$ are used, which is the same as CSGD. The algorithm is shown in Algo.4.1.

Algorithm 4.1 BSGD

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i=1}^{M} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
for *epoch* $= 1, 2, \cdots, K_{max}$ **do**
  for jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
    Random select a column block J_j from sets $\{J_j\}_{j\in[1,N]}$ with replacement
    for ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
      Random select a row block I_i from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      $\mathbf{z}^j_{I_i} = \mathbf{A}^{J_j}_{I_i}(\mathbf{x}_{J_j})$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N} \mathbf{z}^j$
  for the selected I_i and J_j **in parallel do**
    $\hat{\mathbf{g}}^i_{J_j} = (\mathbf{A}^{J_j}_{I_i})^T \mathbf{r}_{I_i}$
  end for
  For all blocks J that have been updated, $\mathbf{g}_J = \sum_{i=1}^{M} \hat{\mathbf{g}}^i_J, \mathbf{x}_J = \mathbf{x}_J + \mu \mathbf{g}_J$
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

The BSGD can also be combined with previously proposed importance sampling strategy, which is named as BSGD-IM, as shown in Algo.4.2

4.3 α and γ selection criteria

For each iteration, the total usage time of $\mathbf{A}^J_I$ is $2\alpha\gamma MN$ and the maximum required parallel nodes is $\alpha\gamma MN$. When α and γ are both 1, then BSGD becomes GD. When $\gamma = 1$ and $\alpha < 1$ and each column block samples the same row blocks each time, then BSGD becomes SAG. An extreme situation is that when $\alpha = \frac{1}{M}, \gamma = \frac{1}{N}$, where the BSGD only uses one $\mathbf{A}^J_I$ for each iteration and thus becomes sequential. It is clear that the computation cost for $\mathbf{g}$ decreases when α, γ decreases. However the variance of $\|\mathbf{g} - \mathbf{g}_{true}\|$ increases at the same time, making the iteration process more stochastic and thus slowing down the reconstruction speed. As a result, how to select α, γ to balance the computation cost and reconstruction speed is of interest here. The selection criteria should reduce the computation cost by using small α and γ while maintaining a fast converging speed. Here the simulation data in Fig.3.13 (i.e. $\mathbf{A} \in \mathbb{R}^{72000\times4096}$) is used.

Algorithm 4.2 BSGD-IM

Initial: The same as Algo.4.1.
for $epoch = 1, 2, \cdots, K_{max}$ **do**
  for jj=1,2,$\cdots$,γN. **In parallel** (J loop) **do**
    Random select a column block J_j from sets $\{J_j\}_{j\in[1,N]}$ with replacement
    for ii=1,2,$\cdots$,αM **In parallel** (I loop) **do**
      Random select a row block I_i from sets $\{I_i\}_{i\in[1,M]}$ with replacement
      Importance sample one sub-detector area from each single projection view.
      All indexes represented by those selected sub-detector areas form the row index set $\hat{I}_t$.
      Each parallel node records its own $\hat{I}_t$ for the later BP procedure.
      $\mathbf{z}^j_{\hat{I}_t} = \mathbf{A}^{J_j}_{\hat{I}_t}\mathbf{x}_{J_j}$
    end for
  end for
  $\mathbf{r} = \mathbf{y} - \sum_{j=1}^{N}\mathbf{z}^j$
  for the selected I_i and J_j **in parallel do**
    $\hat{\mathbf{g}}^i_{J_j} = (\mathbf{A}^{J_j}_{\hat{I}_t})^T\mathbf{r}_{\hat{I}_t}$ (Attention that $\hat{I}_t$ is recorded in each node)
  end for
  For all blocks J that have been updated, $\mathbf{g}_J = \sum_{i=1}^{M}\hat{\mathbf{g}}^i_J, \mathbf{x}_J = \mathbf{x}_J + \mu\mathbf{g}_J$
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

By setting $M = 60, N = 16$, the converging speed with different α, γ settings are shown in Fig.4.1. All the parameters are well tuned and the fastest cases are presented here. It can be seen that for each different γ settings, reducing α under $\frac{1}{2}$ can significantly increase the converging speed. Fig.4.1(f) shows that when reducing α to $\frac{1}{M}$, different γ settings can have slightly different converging speeds but the difference is not obvious. This suggest that in a parallel network where p is far less than MN, reducing α and γ does not cause significant delayed reconstruction speed. In the Fig.4.1 case, $M = 60, N = 16$ means that $\mathbf{A}^J_I \in \mathbf{R}^{1200\times256}$, which is a tall-thin shape. The shape of $\mathbf{A}^J_I$ can also be changed by setting $M = 60, N = 4$. In the Fig.4.2. In the figure, $M = 60, N = 4$ means that $\mathbf{A}^J_I \in \mathbf{R}^{1200\times1024}$, which is a square shape. Simulations show that regardless the shape, reducing α under $\frac{1}{2}$ can still increase the reconstruction speed. Fig.4.2(d) shows that when reducing α to $\frac{1}{M}$, different γ does not show significant differences.

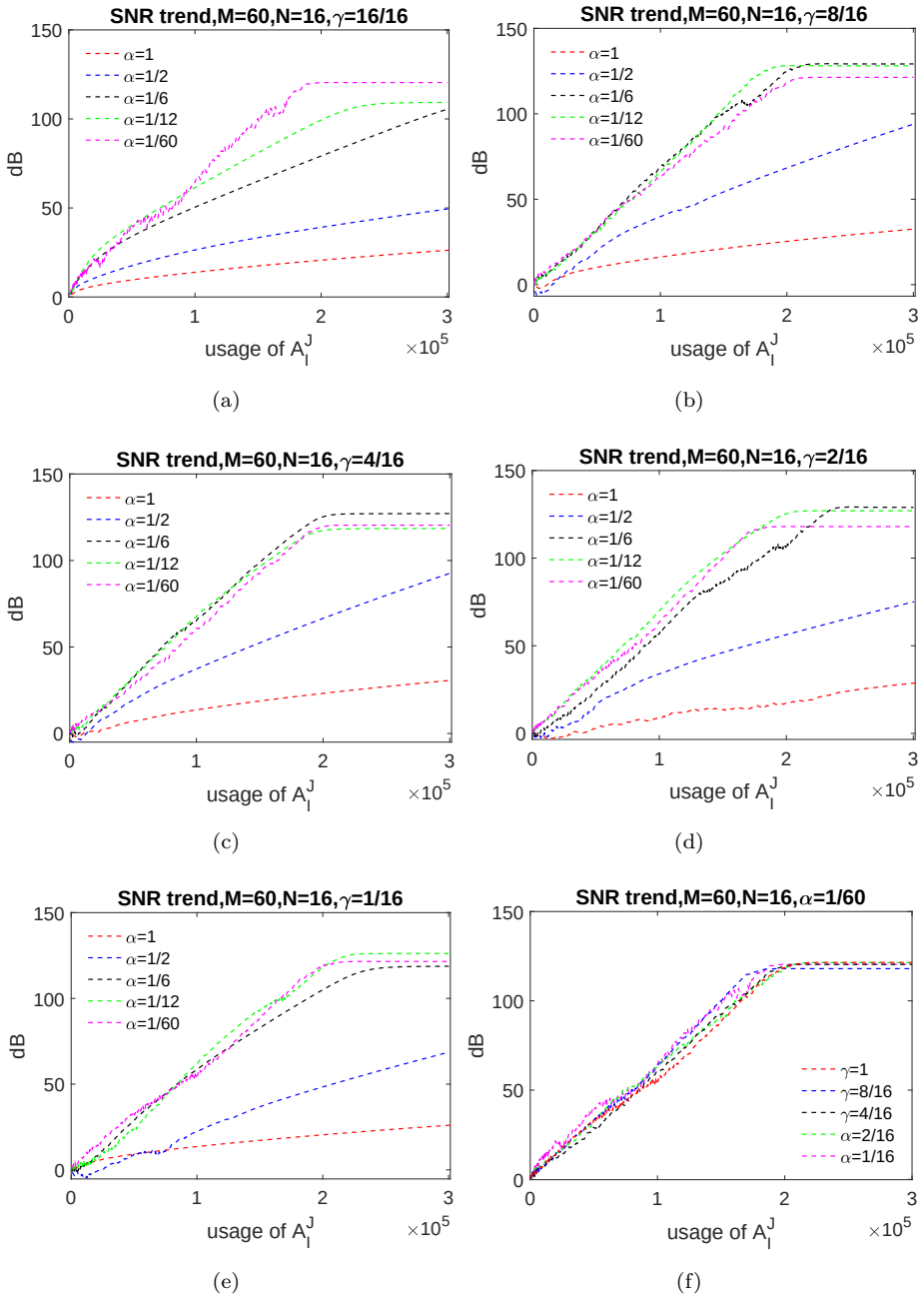


Figure 4.1: The reconstruction speeds under different α, γ settings with a determined partition $M = 60, N = 16$ are compared with each other.

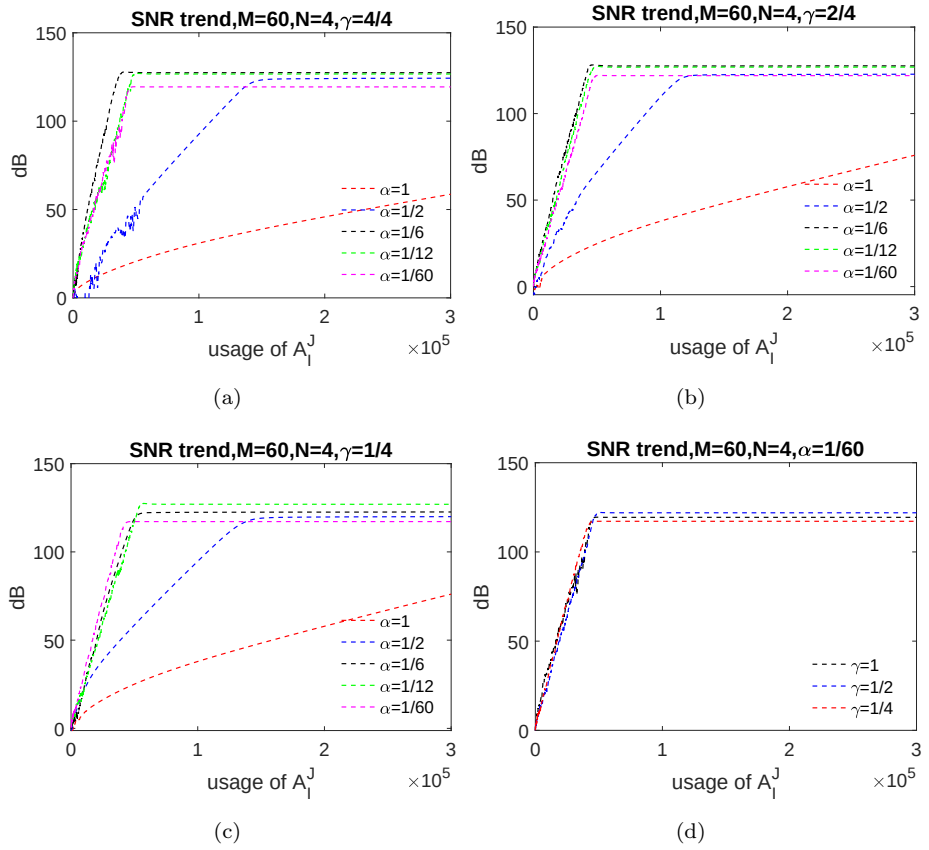


Figure 4.2: Repeated simulations which are similar to Fig.4.1 when the $M = 60, N = 4$.

From the above simulations, the suggested selection criteria is the same as CSGD algorithm, as shown in Eq.3.8. According to the criteria, when p is smaller than N, then α should be reduced to the minimal $\frac{1}{M}$ while keeping γ as large as possible ($\frac{p}{N}$) to cover image blocks as many as possible. The following simulations justify this criteria. For $M = 60, N = 16$ case, assume that p is 8 or 16. For $M = 60, N = 4$ case, assume that number is 2 or 4, then the comparisons of different α, γ are shown in Fig.4.3.

It is important to point out that when p is smaller than N, setting $\gamma = 1$ is not forbidden but is inefficient in 2 cases: 1)The first case is when N, the number of image blocks, cannot be divided by p without remainder. For example, when $N = 6$ but p is 4, then it means that there are 2 "inner loops" for the calculation of $\mathbf{z}_I^j$ and $\hat{\mathbf{g}}_J^i$. Take the calculation of $\mathbf{z}_I^j$ as example. In the 1^{st} inner loop, 4 parallel nodes receive $\{\mathbf{x}_{J_j}\}_{j=1,2,3,4}$. Since $\gamma = 1$, there is a 2^{nd} "inner loop", where 2 parallel node receive $\{\mathbf{x}_{J_j}\}_{j=5,6}$ and the other 2 nodes remains static, which is a waste of computation resources. 2) The second case is when the updates of $\mathbf{r}$, $\mathbf{g}$ and $\mathbf{x}$ are also performed on parallel nodes. For example, assume that $M = N = 4$ and $\alpha = \frac{1}{4}, \gamma = 1$ but p is 2. If $I_1, \{J_j\}_{j=1}^4$ are selected, then the data communication is shown in Fig.4.4, which shows that there are 2 "inner loops" in total for each update of $\mathbf{r}$ and $\mathbf{g}$. The communication cost is larger than the case when $\gamma = \frac{p}{N}$ because an extra transmission on $\mathbf{x}$ block, which happens on (d)

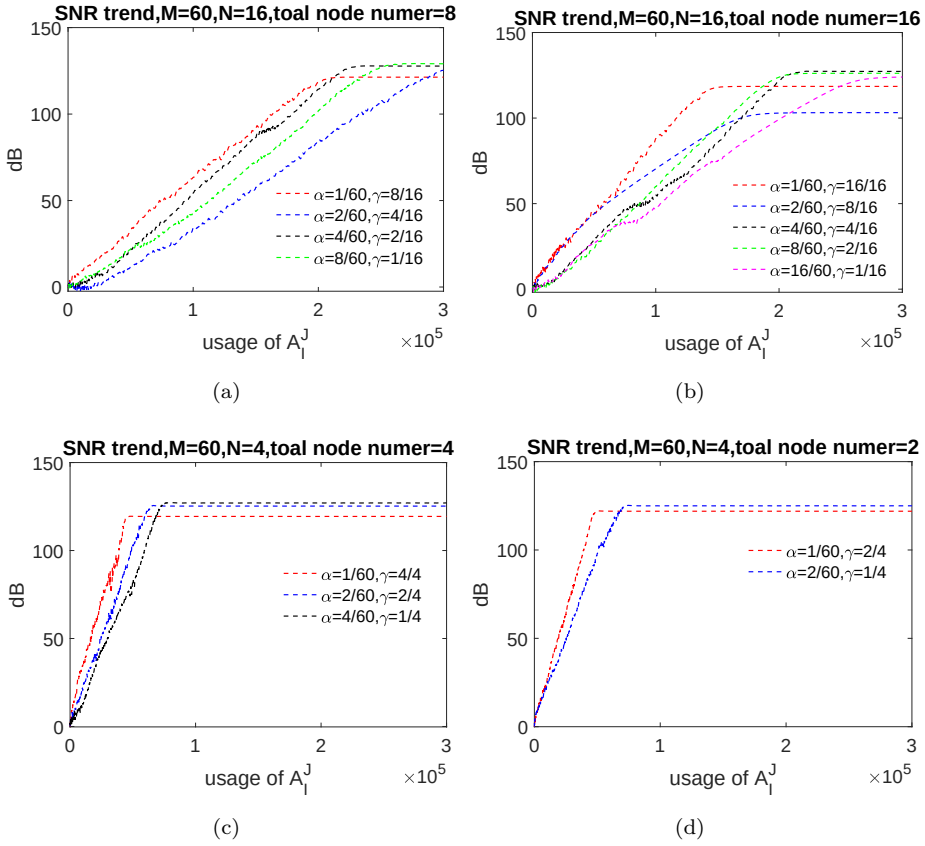
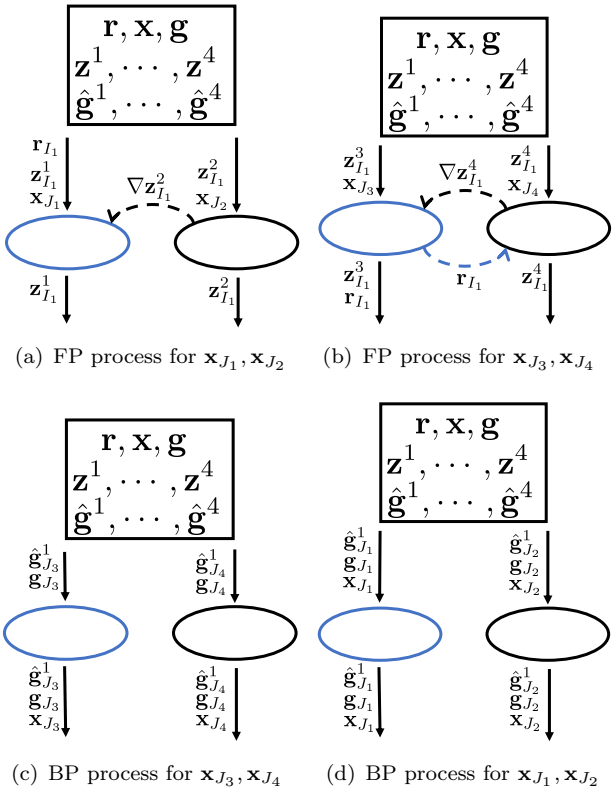


Figure 4.3: Reducing α to $\frac{1}{M}$ and maximizing γ to $\frac{p}{N}$ (red lines)helps a bit to increase the reconstruction speed.

procedure in Fig.4.4. This illustration justifies α, γ selection criteria which encourages to reduce γ as $\frac{p}{N}$ instead of 1.



(a) FP process for $\mathbf{x}_{J_1}, \mathbf{x}_{J_2}$ (b) FP process for $\mathbf{x}_{J_3}, \mathbf{x}_{J_4}$

(c) BP process for $\mathbf{x}_{J_3}, \mathbf{x}_{J_4}$ (d) BP process for $\mathbf{x}_{J_1}, \mathbf{x}_{J_2}$

Figure 4.4: The square box represents the master node and oval boxes represent parallel computation nodes. The blue parallel node is the root node, which is responsible for the date gathering and distribution. If $I_1, \{J_j\}_{j=1}^N$ are selected and there are 2 parallel nodes, then 4 procedures are required: (a) The parallel nodes request $\mathbf{x}_{J_1}, \mathbf{x}_{J_2}$ and old $\mathbf{z}_{I_1}^1, \mathbf{z}_{I_1}^2$. Besides, the root node also requests $\mathbf{r}_{I_1}$. Each computation node calculates difference $\nabla \mathbf{z}_{I_1}^j = \mathbf{z}_{I_1}^j - \mathbf{A}_{I_1}^{J_j} \mathbf{x}_{J_j}, j \in [1, 2]$ and updates $\mathbf{z}_{I_1}^j = \mathbf{A}_{I_1}^{J_j} \mathbf{x}_{J_j}$. The non-root node (black) sends $\nabla \mathbf{z}_{I_1}^2$, and in the root node, $\mathbf{r}_{I_1} = \mathbf{r}_{I_1} + \nabla \mathbf{z}_{I_1}^1 + \nabla \mathbf{z}_{I_1}^2$; Each computation node returns updated $\mathbf{z}_{I_1}^j$. (b) In the 2^{nd} inner-loop, the above procedure is repeated. Attention that $\mathbf{x}_{J_3}, \mathbf{x}_{J_4}$ have to replace $\mathbf{x}_{J_1}, \mathbf{x}_{J_2}$ due to the limited storage capacity. When $\mathbf{r}_{I_1}$ finishes updating, it is broadcasted from the root node to non-root node for the latter gradient calculation. (c) Each computation node requests $\mathbf{g}_{J_j}, \hat{\mathbf{g}}_{J_j}^1, j \in [3, 4]$ and the computation node calculates difference $\nabla \hat{\mathbf{g}}_{J_j}^1 = (\mathbf{A}_{I_1}^{J_j})^T \mathbf{r}_{I_1} - \hat{\mathbf{g}}_{J_j}^1, j \in [3, 4]$ and updates $\hat{\mathbf{g}}_{J_j}^1 = (\mathbf{A}_{I_1}^{J_j})^T \mathbf{r}_{I_1}$. $\mathbf{g}_{J_j}$ is updated as $\mathbf{g}_{J_j} = \mathbf{g}_{J_j} + \nabla \hat{\mathbf{g}}_{J_j}^1$, after updating $\mathbf{g}_{J_j}$, the corresponding $\mathbf{x}_{J_j}$ is updated as $\mathbf{x}_{J_j} = \mathbf{x}_{J_j} + \mu \times \mathbf{g}_{J_j}$. (d) repeat the (c) to update $\mathbf{x}_{J_j}, j \in [1, 2]$. Since $\mathbf{x}_{J_j}, j \in [1, 2]$ are no longer in the computation nodes, the computation nodes have to re-request $\mathbf{x}_{J_j}, j \in [1, 2]$.

4.4 Comparision between CSGD and BSGD

In this section the differences between CSGD and BSGD including the storage demand, parallel computation scheme and their ability to approach the least squares solution are analysed.

To simplify the illustration, assume that at one iteration, I_1 and $J_1, J_2, \cdots, J_{\gamma N}$ are selected, the parallel computation scheme of CSGD and the detailed illustrations are shown in Fig.4.5. The parallel computation scheme of BSGD is more complicated than CSGD.

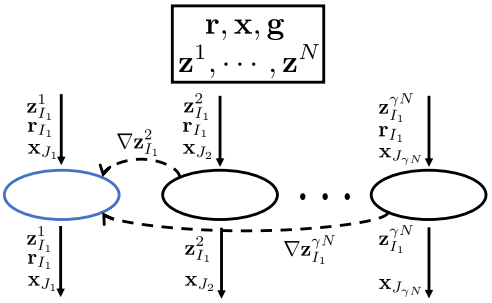
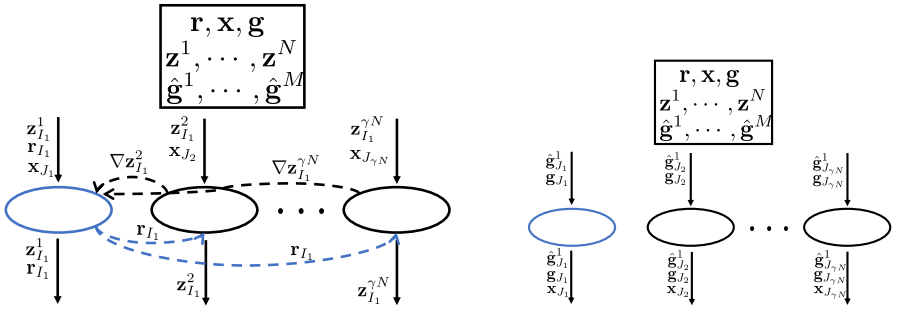


Figure 4.5: Data communication in CSGD. The square box is the master node which stores all necessary variables and the oval boxes are γN parallel computation nodes. The computation nodes request three variable blocks $\mathbf{x}_{J_j}, \mathbf{z}^j_{I_1}, \mathbf{r}_{I_1}$ from master node. After every node finishes the update of $\mathbf{z}^j_{I_1}$ block, $\nabla \mathbf{z}^j_{I_1}$, which is the difference between the old and new updated $\mathbf{z}^j_{I_1}$, are summed up via MPI-reduce (explained in section 6.1) to root node (blue oval node). The output of computation nodes are used to update the corresponding blocks in the master node. Notice that $\mathbf{r}_{I_1}$ is only outputted by the root node.

It contains two master-computation node communications, as illustrated in Fig.4.6.



(a) FP process for $\mathbf{x}_{J_1}, \cdots, \mathbf{x}_{J_{\gamma N}}$ (b) BP process for $\mathbf{x}_{J_1}, \cdots, \mathbf{x}_{J_{\gamma N}}$

Figure 4.6: Data communication in BSGD. (a) The general parallel nodes request two variables $\mathbf{x}_{J_j}, \mathbf{z}^j_{I_1}$ from the master node while the root computation node requests an extra $\mathbf{r}_{I_1}$. $\mathbf{r}_{I_1}$ is updated with the same way as CSGD, followed by a MPI-broadcast (explained in section 6.1) procedure that the root node sends updated $\mathbf{r}_{I_1}$ to the other parallel nodes. The output procedure is also similar to CSGD except that $\mathbf{x}$ block is kept in each node. (b) Each computation node requests $\mathbf{g}_{J_j}, \hat{\mathbf{g}}_{J_j}$ from master node. Notice that since $\alpha = \frac{1}{M}, \gamma = \frac{p}{N}$, there is no inner node communication and $\mathbf{g}_{J_j}$ can be updated parallelly and separately by $\mathbf{g}_{J_j} = \mathbf{g}_{J_j} + \nabla \hat{\mathbf{g}}^1_{J_j}$, where $\nabla \hat{\mathbf{g}}^1_{J_j}$ is the difference between the old and new updated $\hat{\mathbf{g}}^1_{J_j}$ computed within each parallel node.

It should be noticed that the communication scheme of BSGD is the same as other SIRT-type algorithms. Compared with other SIRT-type methods, the previous CSGD

enjoys a simpler communication scheme and thus will enjoy a wider parallel scalability. The detailed discussion on this part is mainly presented in Chapter 6.

The storage demand is another aspect where CSGD outperforms BSGD. Assume that $\tilde{r}$ and $\tilde{c}$ are the size of $\mathbf{y}$ and $\mathbf{x}$ space respectively, Table.4.1 illustrates the storage difference and communication difference between the two methods. BSGD requires more storage

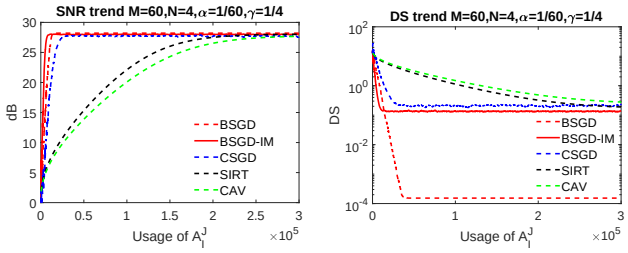
Table 4.1: Comparisons of storage demands and communication overhead for each node between CSGD and BSGD

		CSGD	BSGD
total storage		$(N+1)\tilde{r}+2\tilde{c}$	$(N+1)\tilde{r}+(M+2)\tilde{c}$
Master-Computation nodes communication	root request	$2\frac{\tilde{r}}{M}+\frac{\tilde{c}}{N}$	$2\frac{\tilde{r}}{M}+3\frac{\tilde{c}}{N}$
	root push	$2\frac{\tilde{r}}{M}+\frac{\tilde{c}}{N}$	$2\frac{\tilde{r}}{M}+3\frac{\tilde{c}}{N}$
	other request	$2\frac{\tilde{r}}{M}+\frac{\tilde{c}}{N}$	$\frac{\tilde{r}}{M}+3\frac{\tilde{c}}{N}$
	other push	$\frac{\tilde{r}}{M}+\frac{\tilde{c}}{N}$	$\frac{\tilde{r}}{M}+3\frac{\tilde{c}}{N}$
Computation nodes inner communication	root MPI-reduce	$(\gamma N-1)\frac{\tilde{r}}{M}$	$(\gamma N-1)\frac{\tilde{r}}{M}$
	root broadcast	0	$\frac{\tilde{r}}{M}$

space and communication overhead compared to CSGD. This is mainly caused by the new introduced variable $\{\hat{\mathbf{g}}^i\}_{i=1}^M$. However, the new introduced variable can gradually reduce the error variance of the estimated update direction, making the iteration faster and closer to the least square solution than CSGD and some other existing methods like CAV and SIRT. To verify the advantage of BSGD, the dataset used in Fig.3.13 is adopted again and is blurred by Gaussian-type noise, with the SNR of each projection set to 31.1dB. A term DS is introduced to reflect the distance to the least square solution. It is defined as:

$$DS = \|\mathbf{x}_{lsq} - \mathbf{x}^k\|, \tag{4.1}$$

where $\mathbf{x}_{lsq}, \mathbf{x}^k$ are the least square solution and the reconstructed image vector at k^{th} epoch. Simulation results are shown in Fig.4.7. All parameters in each method are well tuned to ensure the fastest reconstruction speed. We see that BSGD not only approaches the least square solution, it also shows a faster reconstruction speed compared to CSGD, SIRT and CAV. The figure also show that BSGD-IM is of the fastest reconstruction speed, achieving the SNR limit within the least usage of $\mathbf{A}_I^J$. However, it does not reach the least square solution. This is because that the BP procedure in line 15 of Algo.4.2 is an inaccurate calculation procedure. To be more specific, in BSGD, the $\hat{\mathbf{g}}^i_{J_j} = (\mathbf{A}^{J_j}_{I_i})^T\mathbf{r}_{I_i}$. In BSGD-IM, however, the $\hat{\mathbf{g}}^i_{J_j} = (\mathbf{A}^{J_j}_{\hat{I}_t})^T\mathbf{r}_{\hat{I}_t}$, where $\hat{I}_t$ is a subset of I_i. This approximated calculation inevitably introduces error and thus cannot enable the sum of $\hat{g}^i$ to approach the true gradient direction $\mathbf{g}_{true}$, making the final solution cannot approach the least square solution.



(a) SNR trend (b) DS trend

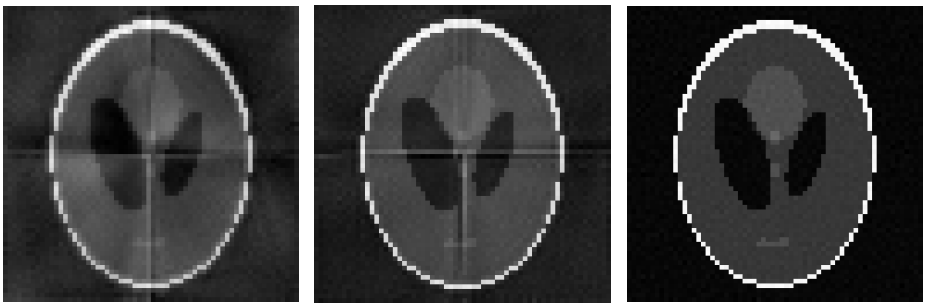
Figure 4.7: The projections are blurred by a white Gaussian noise whose variance $\sigma = 0.2$. In contrast to BSGD, SIRT, CAV and CSGD do not achieve the least square solution and are slower in terms of the reconstruction speed than new proposed BSGD.

The reconstructed images are presented in Fig.4.8. From the figure it can be seen that both CSGD and BSGD generate artifacts at the border of sub-image. However, this artifacts gradually disappear as iteration goes on and the quality of final reconstructed images is good enough to present the inner details of the object, as SIRT methods does. The BSGD-IM also generates artifacts. However due to the fastest reconstruction speed, the artifacts already disappears after 5000 matrix-vector multiplications. All methods finally present clear enough inner details of the object, showing the effectiveness of the proposed CSGD, BSGD and BSGD-IM in CT reconstruction. Furthermore, by comparing the reconstructed images, SIRT and CSGD can present clear results after 50000 projections, BSGD and BSGD-IM can present satisfying results after 10000 and 5000 projections. As a result, the BSGD-IM is of fastest reconstruction speed, the BSGD ranks the second place, the CSGD and SIRT approximately has the same reconstruction speed. It should be pointed out that the image quality is estimated by the SNR of the image, i.e. using Fig.4.7(a) instead of the DS trend. Although the BSGD-IM does not converge to the least square solution, the quality of reconstructed image is not influenced by this property. The SNR reflects the distance between the reconstructed image and the true image vector whilst the DS reflects the data fidelity. Unable to converge to the least square solution means that the data fidelity cannot be minimised, but it does not influence the distance to the true image vector. This is the reason why the BSGD-IM does not approach to the least square solution closer than BSGD but the reconstruction speed is faster and the final reconstruction results have negligible differences between the other methods.

4.5 Best partition of M,N

This section discusses the optimal partition on $\mathbf{y}$ and $\mathbf{x}$ space. Assume that the storage amount for each computation node is G and $G < \tilde{c} < \tilde{r}$, it is clear that there is an optimal M and N setting that makes the total storage amount in the master node

(a) SIRT (b) SIRT (c) SIRT



(d) CSGD (e) CSGD (f) CSGD

(g) BSGD (h) BSGD (i) BSGD

(j) BSGD-IM (k) BSGD-IM (l) BSGD-IM

Figure 4.8: The first column is 5,000 "FP+BP" projections while the second and the third are 10,000 and 50,000 projections respectively. BSGD and CSGD finally reconstruct the images with high quality, despite of the margin artefacts at initial iterations. The BSGD-IM is of the fastest reconstruction speed, providing a clear reconstruction result within only 5,000 FP+BP projections. The BSGD is the second fastest method, which provides clear reconstructed image within 10,000 usage of $\mathbf{A}_I^J$. Reconstructed images by CAV are similar to SIRT and thus are omitted here.

(i.e. square box in Fig.4.6) minimal while enabling the data blocks to be small enough to fit in the computation nodes. According to Table.4.1, the total storage demand is $(N+1)\tilde{r}+(M+2)\tilde{c}$. According to Fig.4.6, the computation nodes first store $\mathbf{z}_I^j, \mathbf{r}_I, \mathbf{x}_J$ and then push $\mathbf{z}_I^j$ back to the master node while maintaining the other two data blocks. At this stage, it means that the computation node must be large enough to store $\mathbf{z}_{I_i}^j, \mathbf{r}_{I_i}, \mathbf{x}_{J_j}$ and $\nabla\mathbf{z}_{I_i}^j$. Then the computation nodes request and store $\hat{\mathbf{g}}_{J_j}^i$ and $\mathbf{g}_{J_j}$. As a result, G must also be large enough to store $\mathbf{r}_{I_i}, \mathbf{x}_{J_j}, \hat{\mathbf{g}}_{J_j}^i, \mathbf{g}_{J_j}$ and $\nabla\hat{\mathbf{g}}_{J_j}^i$.

The total constrained optimization then becomes:

$$\begin{aligned} \min \quad & (N+1)\tilde{r} + (M+2)\tilde{c}, \\ \text{s.t.} \quad & 3\frac{\tilde{r}}{M} + \frac{\tilde{c}}{N} \leqslant G \\ & \frac{\tilde{r}}{M} + 4\frac{\tilde{c}}{N} \leqslant G \\ & 1 - N \leqslant 0 \\ & 1 - M \leqslant 0. \end{aligned} \tag{4.2}$$

The lagrangian equation of this constrained problem can be expressed as

$$\mathcal{L} = (N+1)\tilde{r}+(M+2)\tilde{c}+\mu_1(3\frac{\tilde{r}}{M}+\frac{\tilde{c}}{N}-G)+\mu_2(\frac{\tilde{r}}{M}+4\frac{\tilde{c}}{N}-G)+\mu_3(1-N)+\mu_4(1-M). \tag{4.3}$$

The Karush–Kuhn–Tucker (KKT) conditions can be expressed as:

$$\begin{aligned} &(a)\tilde{r} - \mu_1\tilde{c}\frac{1}{N^2} - 4\mu_2\tilde{c}\frac{1}{N^2} - \mu_3 = 0 \\ &(b)\tilde{c} - 3\mu_1\tilde{r}\frac{1}{M^2} - \mu_2\tilde{r}\frac{1}{M^2} - \mu_4 = 0 \\ &(c)3\frac{\tilde{r}}{M} + \frac{\tilde{c}}{N} - G \leqslant 0 \\ &(d)\frac{\tilde{r}}{M} + 4\frac{\tilde{c}}{N} - G \leqslant 0 \\ &(e)1 - M \leqslant 0 \\ &(f)1 - N \leqslant 0 \\ &(g)\mu_1(3\frac{\tilde{r}}{M} + \frac{\tilde{c}}{N} - G) = 0 \\ &(h)\mu_2(\frac{\tilde{r}}{M} + 4\frac{\tilde{c}}{N} - G) = 0 \\ &(i)\mu_3(1 - N) = 0 \\ &(j)\mu_4(1 - M) = 0 \\ &(k)\mu_1, \mu_2, \mu_3, \mu_4 \geqslant 0 \end{aligned} \tag{4.4}$$

In fact, μ_3, μ_4 should both be 0, otherwise M or N is 1, which violates the KKT condition in (c) and (d).

If $\mu_1 = 0$ and $\mu_2 \neq 0$, according to $(a),(b),(h)$, $\frac{\tilde{r}}{M} = 2\frac{\tilde{c}}{N}$, and $M = 3\frac{\tilde{r}}{G}, N = 6\frac{\tilde{c}}{G}$. Similarly, if $\mu_2 = 0$ and $\mu_1 \neq 0$, according to $(a),(b),(g)$, $\sqrt{3}\frac{\tilde{r}}{M} = \frac{\tilde{c}}{N}$, and $M = (3+\sqrt{3})\frac{\tilde{r}}{G}, N = (\sqrt{3}+1)\frac{\tilde{c}}{G}$. However, these solutions violate either (c) or (d). This means that μ_1, μ_2 should both be non-zero. According to (g) and (h), $M = \frac{11}{3}\frac{\tilde{r}}{G}, N = \frac{11}{2}\frac{\tilde{c}}{G}$. Since M, N should both be integer then optimal partition on $\mathbf{x}, \mathbf{y}$ space is shown in Eq.4.5.

$$\begin{aligned} M &= \lceil \frac{11\tilde{r}}{3G} \rceil \\ N &= \lceil \frac{11\tilde{c}}{2G} \rceil. \end{aligned} \tag{4.5}$$

4.6 Automatic parameter tuning

In this section, an automatic parameter tuning strategy is proposed to accelerate the reconstruction speed when the step-length μ is not set properly. Broadly speaking, up to a limit, increasing μ increases reconstruction speed. However, in practice, it is difficult to determine the upper limit. As a result, in realistic large scale tomographic reconstructions, instead of using a fixed step-length μ, an automatic parameter tuning approach is adopted. Parameter tuning is not a new concept in machine learning and optimization. For example, the hypergradient descentAtilim Gunes et al. (2017) or the Barzilai-Borwein (BB) method (Conghui et al., 2016) can be used for SGD or SVRG. However these methods are not directly applicable to BSGD, as they require updating all of $\mathbf{x}_{est}$ in each iteration. Furthermore, BSGD uses dummy variables $\mathbf{z}$ to store information about previous $\mathbf{x}_{est}$. Due to this, the stochastic gradient calculated by BSGD is much noisier than the estimate obtained by traditional stochastic gradient methods. An automatic parameter tuning method is proposed here which is different from the BB method or hypergradient descent method. It only exploits the parameters generated during the iteration process: the residual $\mathbf{r}$ and update direction $\mathbf{g}$, as shown in Algo.4.3.

This automatic parameter tuning is applied after f epochs. It tests whether $\|\mathbf{r}\|$ is continually decreased during the past $2f$ epochs, in which case μ is increased by $1+\epsilon$. To reduce μ, using a similar condition on $\mathbf{r}$ alone (i.e. line 8 in Algo.4.3, named as "criteria 1") was not found to be sufficient to ensure a high accuracy solution. An additional criteria (line 9 in Algo.4.3, named as "criteria 2") is thus adopted. The criteria 2 is motivated by the general parameter tuning methods that compute inner-products between adjacent gradients and determine to increase or decrease μ according to the positivity of the inner-product (Atilim Gunes et al., 2017; Plakhov and Cruz, 2004). The proposed automatic parameter tuning trick thus compares the inner-products of two "virtual" gradients. To do this, several stochastic gradients during a period of several epochs (f epochs in Algo.4.3) are accumulated to compute an effective update direction (EUD) $\overline{\mathbf{g}}$ to reduce the stochastic error variance. It is observed that when

Algorithm 4.3 Automatic μ tunning strategy

1: t_1, t_2, ϵ and δ are positive constants, $f = M$.
2: At each k^{th} epoch where $mod(k, f) == 0$, sum up all $\mathbf{g}$ in the past f epoch as an effective update direction (EUD) $\overline{\mathbf{g}}^{k/f}$.
3: Calculate the inner product between two consecutive $\overline{\mathbf{g}}$ as $\theta^{k/f} = \frac{(\overline{\mathbf{g}}^{k/f})^T \overline{\mathbf{g}}^{k/f-1}}{\|\overline{\mathbf{g}}^{k/f}\| \|\overline{\mathbf{g}}^{k/f-1}\|}$.
4: **if** mod(k,f)==0&$k > f$ **then**
5: **if** $\|\mathbf{r}\|^k < \|\mathbf{r}\|^{k-f} < \|\mathbf{r}\|^{k-2f}$ and $\theta^{k/f} > t_2$ **then**
6: $\mu = (1 + \epsilon)\mu$
7: **end if**
8: **if** $\|\mathbf{r}\|^k > \|\mathbf{r}\|^{k-f} > \|\mathbf{r}\|^{k-2f}$ **then**
9: **if** $\mid \theta^{k/f} - \theta^{k/f-1} \mid > t_1$ or $\theta^{k/f} < t_2$ **then**
10: $\mu = (1 - \delta)\mu$
11: **end if**
12: **end if**
13: **end if**

BSGD approaches a stable solution with a properly fixed μ, then the change of two adjacent EUDs does not vary significantly. On the contrary, these two directions vary significantly when BSGD suffers from oscillatory behaviour or an increase in the norm of $\mathbf{r}$. This is because that BSGD uses some old values $\mathbf{z}/\hat{\mathbf{g}}$ in the calculation of each update. To ensure the convergence, the changes in $\mathbf{x}$ among a period of epochs is not allowed to be too large, then these old values for $\mathbf{z}/\hat{\mathbf{g}}$ are good approximations to the current values. Due to the small movement of $\mathbf{x}$ during $2f$ epochs, the two EUDs, should also be similar to each other and the θ, which represents the cosine angle of two EUDs is always close to 1 in a converging iterations (i.e. t_2 is close to 1 and t_1, which reflects the change of θ, is close to 0). If not, it means that the step-length is too big and the iteration is likely to diverge.

In this section, the simulation data all uses the one in Fig.3.13. The variance of gaussian type noise is 0.2, making the SNR of $\mathbf{y}$ 31.1 dB. When $\delta = 0.4$ and $\epsilon = 0.04$, the results are shown in Fig.4.9. In the figure, blue lines mean step length for BSGD (fixed μ form) is always 2×10^{-7}. For BSGD with automatic parameter tuning, however, it means that the initial step length is 2×10^{-7} and its value will be tuned in future's iteration. The other colors have similar meanings. Results in Fig.4.9(a)-(d) demonstrate that automatic tuning (solid lines) provides faster reconstruction speed than the original method when initial step-length μ_0 is set improperly (dashed lines). When the initial step length is set as 2×10^{-5}, the original BSGD diverges whilst the BSGD with automatic parameter tuning generates a converging iterations by gradually tuning the step length to a proper range, as shown in Fig.4.9(e)-(f). Despite that, the automatic parameter cannot guarantee that it converges to the least square solution, the SNR trend empirically shows that it reconstructs the image to the same level as BSGD using constant step-lengths. Only two α, γ cases are provided here and similar results (omitted for brevity) are obtained for other values of M, N, α and γ.

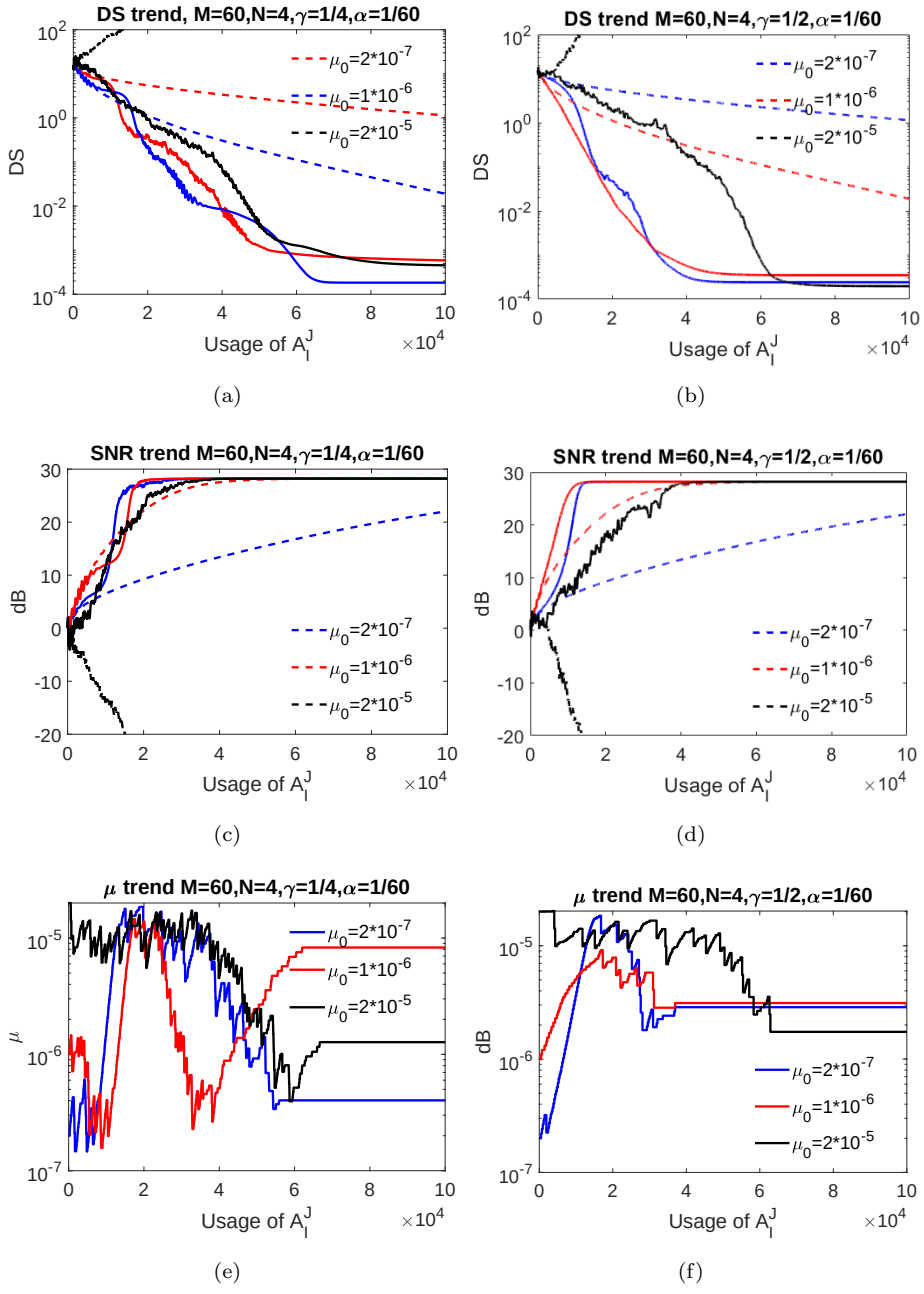
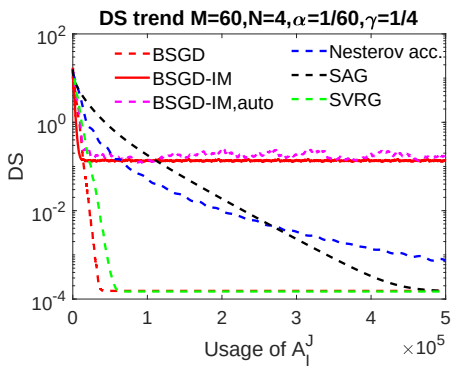


Figure 4.9: BSGD (dashed) vs BSGD with parameter tuning (solid), using $t_1 = 0.1, t_2 = 0.7, \epsilon = 0.04, \delta = 0.4$. It can be seen that for $\mu_0 = 2 \times 10^{-5}$, the original BSGD does not converge but automatic parameter tuning, when initialised with $\mu_0 = 2 \times 10^{-5}$, can adapt the step-lengths into a range leading to stable iterations.

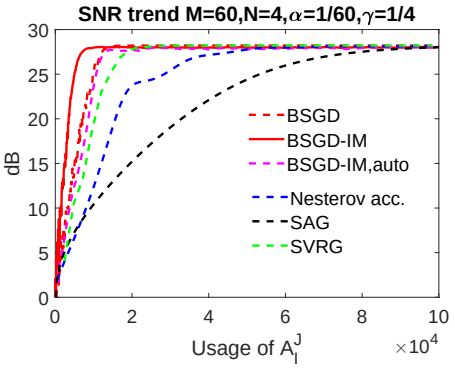
For both increasing and decreasing μ, the frequency of parameter changes is $f = M$ epochs rather than 1 epoch. One reason is that the high stochastic noise in the gradient updates can be reduced after a period of epochs. Another reason is that the calculation of $\|\mathbf{r}\|$ can be time consuming when the size of $\mathbf{r}$ is large and reducing the frequency of computation on $\|\mathbf{r}\|$ is beneficial to save the reconstruction time. It has been experimentally validated that setting the test frequency f to M leads to a good compromise between increased computational demand and improved overall reconstruction speed. However, the setting of frequency can be flexible. For example, the frequency can be set as a value with which the whole $\mathbf{x}$ space has just been updated at least once. If N can divide γ without remainder, and the set $\{J_j\}_{j=1}^{N}$ is sampled without replacement, then $f = \frac{1}{\gamma}$ is also feasible.

4.7 Comparison to other methods

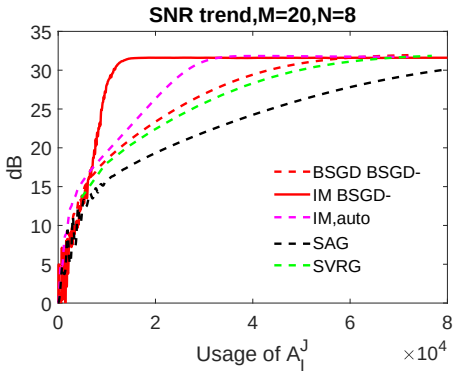
In section 4.4, the comparison of CSGD and BSGD shows that BSGD has a faster reconstruction speed even though BSGD has more complex communication schemes. In this section BSGD is compared to other SIRT-type methods including GD, GD with Nesterov acceleration, SAG and SVRG methods. They all have the same communication scheme and the only difference is that the other SIRT-type methods unavoidably update all of $\mathbf{x}$ in each step, thus when $N > p$, the communication overhead, as well as the usage of $\mathbf{A}_I^J$, is larger than BSGD. In this section, both 2D and 3D simulations are presented. The scanning geometries are the same as in Fig.3.13 and Fig.3.25. The comparisons are shown in Fig.4.10. All simulations, except the automatic parameter setting case, are well tuned to ensure the fastest reconstruction speed. It can be seen that the reconstruction speed of automatic parameter tuning is slightly slower because of the non-optimally chosen small initial step-lengths. For well-tuned parameters, BSGD shows similar reconstruction speed to SVRG. However, according to Fig.4.4, when p is smaller than N, then BSGD has a simpler communication scheme compared to SVRG. This is because that each parallel node only updates one image blocks. The BSGD-IM is the fastest among all reference methods. The detector partitioning is the same as discussed in Fig.3.20(b) and Fig.3.24. It clearly shows that BSGD-IM, which is the combination of BSGD and the importance sampling trick, is the fastest method. One drawback of BSGD-IM is that it does not converge to the least square solution, but the final SNR limit of BSGD-IM is similar to the other methods, which means that the final solution's accuracy is similar to the other methods.



(a) 2D,SNR of $\mathbf{y}$ is 31.1dB



(b) 2D,SNR of $\mathbf{y}$ is 31.1dB



(c) 3D,SNR of $\mathbf{y}$ is 36.1dB

Figure 4.10: Comparison of BSGD with other SIRT-type methods for a 2D (top) and 3D (bottom) setup. The projection data $\mathbf{y}$ are blurred by Gaussian noise. In 3D case, due to the large computation complexity on the least square solution, DS trend along with usage of $\mathbf{A}_I^J$ is omitted. For 2D and 3D case the automatic parameter setting methods use initial step-length 1×10^{-6}. The step lengths in the other methods have been well selected to ensure the fastest reconstruction speeds.

4.8 Further trials on accelerating BSGD

Section 4.6 presents an automatic parameter tuning strategy to accelerate the reconstruction speed of BSGD when its step length μ is set improperly small or large. However, this automatic parameter tuning does not works well if the μ has been well tuned through repeated trials in simulations. In this section, further trials on acceleration the BSGD when μ is well tuned is presented.

One of the most famous and widely applied acceleration tricks is Nesterov's acceleration, as demonstrated in Algo.2.1. It can be seen that Nesterov's acceleration performs a simple step of gradient descent to go from $\mathbf{x}^k$ to $\tilde{\mathbf{x}}^{k+1}$, and then it moves further than $\tilde{\mathbf{x}}^{k+1}$ in the direction given by the previous point $\tilde{\mathbf{x}}^k$.

However, this momentum acceleration is dangerous for stochastic SIRT-type methods. During iterations, each update direction is not an accurate direction and if this stochastic direction is added to the momentum and moves further along this direction, it may hurt the performance because it leads to error accumulation. Simulations also verify that when BSGD is combined with Nesterov's acceleration methods, no accelerated reconstruction speeds can be obtained.

Recently, the "Katyusha" acceleration method was proposed (Allen-Zhu, 2017), which is the first acceleration method for stochastic gradient descent methods. The main ingredient of this kind of acceleration is "Katyusha momentum", a novel "negative momentum" on top of Nesterov's momentum. It can be incorporated into a variance-reduction based algorithm and speed it up. Since previous simulations have verified that the BSGD has the same properties with variance-reduction based algorithms, it is worthy for BSGD to give Katyusha a hug.

The classical Katyusha is combined with SVRG and the momentum step is applied after every single stochastic step. The SVRG-Katyusha is shown as Algo.4.4. It can

Algorithm 4.4 Katyusha acceleration on SVRG

Initialization: $\mathbf{w}^1 = \mathbf{v}^1 = \tilde{\mathbf{x}}^1 = \mathbf{x}^1$
for $k = 1, 2, \cdots, K_{max}$ **do**
  $\mathbf{g}^k = \nabla f(\tilde{\mathbf{x}}^k)$
  for $t = 1, 2, \cdots, f$ **do**
    $s = (k-1)f + t$
    $\mathbf{x}^{s+1} = \tau_1 \mathbf{v}^s + 0.5\tilde{\mathbf{x}}^k + (1 - \tau_1 - 0.5)\mathbf{w}^s$
    $\tilde{\nabla}^{s+1} = \mathbf{g}^k + M\nabla f_{I_i}(\mathbf{x}^{s+1}) - M\nabla f_{I_i}(\tilde{\mathbf{x}}^k)$.
    $\mathbf{v}^{s+1} = \mathbf{v}^s - \alpha\tilde{\nabla}^{s+1}$
    $\mathbf{w}^{s+1} = \mathbf{x}^{s+1} + \tau_1(\mathbf{v}^{s+1} - \mathbf{v}^s)$
  end for
  $\tilde{\mathbf{x}}^{k+1} = \frac{1}{f}(\sum_{t=1}^{f} \mathbf{w}^{(k-1)f+t})$
end for

be seen that after a full gradient calculation, the momentum step is applied after each inner-iteration. However, BSGD was experimentally observed to be not suitable for this kind of acceleration. This is because some $\mathbf{z}^j_{I_i}/\hat{\mathbf{g}}^i_{J_j}$ inevitably store stale forward/back projection information and thus BSGD cannot afford frequent momentum shifts. For example, for the current k^{th} epoch, due to $\alpha, \gamma < 1$, some $\mathbf{z}^j_{I_i}$ inevitably store results for $\mathbf{A}^{J_j}_{I_i}\mathbf{x}^{k-\tau}_{J_j}$, where τ can be 1,2 or even larger integer, which is influenced by α and γ. To be more specific, let us start with the analysis on the original BSGD method. At the k^{th} iteration, $\mathbf{x}^{k+1}_{J_j} = \mathbf{x}^k_{J_j} + \mu\mathbf{g}^k_{J_j}$. $\mathbf{z}^j_{I_i}$ and $\hat{\mathbf{g}}^i_{J_j}$ store the forward and back projection information of $\mathbf{x}^k, \mathbf{x}^{k-1}, \cdots, \mathbf{x}^{k-\tau}$, here τ is a delayed coefficient influenced by α, γ. For example, when $\alpha = \gamma = 1$, $\tau = 0$, when $\alpha = \frac{1}{M}, \gamma = \frac{1}{N}$, and if I_i, J_j are sampled without replacement, $\tau = MN - 1$. With proper step-lengths, the information stored in $\mathbf{z}^j_{I_i}$ and $\hat{\mathbf{g}}^i_{J_j}$ is similar to $\mathbf{A}^{J_j}_{I_i}\mathbf{x}^k_{J_j}$ and $(\mathbf{A}^{J_j}_{I_i})^T(\mathbf{y}_{I_i} - \sum_{j=1}^{N}\mathbf{A}^{J_j}_{I_i}\mathbf{x}^k_{J_j})$, thus making the calculated $\mathbf{g}^k$ similar to true gradient $\mathbf{g}^k_{true}$ on expectation. As a result, when this

process is repeatedly applied to $(k+1)^{th}, (k+2)^{th}$ iterations, a converging trend is obtained. Now consider that when Katyusha acceleration is applied after every single $\mathbf{x}$ update. If $\mathbf{x}^{k+1}$ is further moved by a momentum step to $\tilde{\mathbf{x}}^{k+1}$ after k^{th} iteration, it can be imagined that the average distance between $\tilde{\mathbf{x}}^{k+1}$ and $\mathbf{x}^k, \mathbf{x}^{k-1}, \cdots, \mathbf{x}^{k-\tau}$ (defined as d in this section) becomes larger than that between $\mathbf{x}^{k+1}$ and $\mathbf{x}^k, \mathbf{x}^{k-1}, \cdots, \mathbf{x}^{k-\tau}$. In the next $(k+2)^{th}$ iteration, the information stored in $\mathbf{z}^j_{I_i}, \hat{\mathbf{g}}^i_{J_j}$ may still be able to approximate $\mathbf{A}^{J_j}_{I_i}\tilde{\mathbf{x}}^{k+1}_{J_j}$ and $(\mathbf{A}^{J_j}_{I_i})^T(\mathbf{y}_{I_i} - \sum_{j=1}^{N}\mathbf{A}^{J_j}_{I_i}\tilde{\mathbf{x}}^{k+1}_{J_j})$. However if this acceleration is applied after every $\mathbf{x}$ update, the distance d will continue increasing and there will be a point that the information stored in $\mathbf{z}^j_{I_i}, \hat{\mathbf{g}}^i_{J_j}$ is too old to approximate $\mathbf{A}^{J_j}_{I_i}\tilde{\mathbf{x}}_{J_j}$ and $(\mathbf{A}^{J_j}_{I_i})^T(\mathbf{y}_{I_i} - \sum_{j=1}^{N}\mathbf{A}^{J_j}_{I_i}\tilde{\mathbf{x}}_{J_j})$. As a result, applying the original Katyusha acceleration, which manually moves $\mathbf{x}^{k+1}$ a bit further away from $\mathbf{x}^k$ after every iteration is not a good idea. In 2008, a modified Katyusha was proposed, which is named KtyushaX^s (Allen-Zhu, 2018). Its main difference with classical Katyusha is that the KatyushaX^s applies a momentum step after a period of iterations. The KtyushaX^s combined with SVRG is shown in Algo.4.5. where the SVRG^{1ep} is one epoch process of SVRG and is

Algorithm 4.5 SVRG with KatyushaX^s

1: Initialization: Partition data and $\mathbf{A}$ into M row blocks. $f = 2M$. Determine the maximum allowed iteration number K. Initialization: $\tilde{\mathbf{x}}^0 = \tilde{\mathbf{x}}^1 = \mathbf{x}^1$
2: **for** $k = 1, 2, ..., K_{max}$ **do**
3: $\quad \mathbf{x}^{k+1} = \frac{\frac{3}{2}\tilde{\mathbf{x}}^k + \frac{1}{2}\mathbf{x}^k - (1-\tau)\tilde{\mathbf{x}}^{k-1}}{1+\tau}$
4: $\quad \tilde{\mathbf{x}}^{k+1} = SVRG^{1ep}(\mathbf{x}^{k+1}, \mu, M, f)$
5: **end for**
6: $\mathbf{x}_{solution} = \mathbf{x}^{K_{max}+1}$

defined in Algo.4.6.

Algorithm 4.6 $\tilde{\mathbf{x}} = \text{SVRG}^{1ep}(\mathbf{x}, \mu, M, f)$

1: $\tilde{\mathbf{x}} = \mathbf{x}$
2: $\mathbf{g} = \nabla f(\mathbf{x})$
3: **for** $t = 1, 2, \cdots, f$ **do**
4: $\quad$ Randomly select an i from $[1, M]$
5: $\quad \tilde{\nabla}_t = \mathbf{g} + M\nabla f_{I_i}(\tilde{\mathbf{x}}) - M\nabla f_{I_i}(\mathbf{x})$
6: $\quad \tilde{\mathbf{x}} = \tilde{\mathbf{x}} - \mu\tilde{\nabla}_t$
7: **end for**

BSGD and BSGD-IM can be combined with KatyushaX^s. To simplify the algorithm description, Algo.4.1 is then expressed as Algo.4.7. where BSGD^{1ep} is a "black box" function with input $\mathbf{x}, \mu, \alpha, \gamma, \mathbf{z}, \mathbf{g}$, performing line 3-15 in Algo.4.1. The output of this function is the updated variables $\mathbf{x}, \mathbf{z}, \mathbf{g}$. Inspired by Algo.4.5, the BSGD-KatyushaX^s algorithm is shown in Algo.4.8. Combining BSGD-IM with KatyushaX^s is similar to Algo.4.8 and is not shown here.

Algorithm 4.7 BSGD, simplified expression

1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i=1}^{M} = \mathbf{0}$, and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
2: **for** *epoch* $= 1, 2, ..., K_{max}$ **do**
3: $[\mathbf{x},\mathbf{z},\mathbf{g}]$=BSGD$^{1ep}(\mathbf{x},\mu,\alpha,\gamma,\mathbf{z},\mathbf{g})$
4: **end for**
5: $\mathbf{x}_{solution} = \mathbf{x}$

Algorithm 4.8 BSGD-KatyushaX^s

1: $\tilde{\mathbf{x}} = \mathbf{x}_{tem} = \mathbf{x} = \mathbf{0}$. f is the frequency of applying a momentum step. $\tau \in (0, 0.5)$ is a constant.
2: $\tilde{\mathbf{x}}^0$=$\tilde{\mathbf{x}}^1$=$\mathbf{x}^1$
3: **for** $k = 1, 2, ..., K_{max}$ **do**
4: $\mathbf{x}^{k+1} = \frac{\frac{3}{2}\tilde{\mathbf{x}}^k+\frac{1}{2}\mathbf{x}^k-(1-\tau)\tilde{\mathbf{x}}^{k-1}}{1+\tau}$
5: $\mathbf{x}_{tem} = \mathbf{x}^{k+1}$
6: **for** $t = 1, 2, \cdots, f$ **do**
7: $[\mathbf{x}_{tem},\mathbf{z},\mathbf{g}]$=BSGD$^{1ep}(\mathbf{x}_{tem},\mu,\alpha,\gamma,\mathbf{z},\mathbf{g})$
8: **end for**
9: $\tilde{\mathbf{x}}^{k+1} = \mathbf{x}_{tem}$
10: **end for**
11: $\mathbf{x}_{solution} = \tilde{\mathbf{x}}^{K_{max}+1}$

Similar to SVRG with KatyushaX^s, when $\tau = 0.5$, the BSGD-KatyushaX^s is the same as BSGD as $\tilde{\mathbf{x}}^k = \mathbf{x}^{k+1}$. As a result, in simulations τ is tuned under 0.5. The BSGD-KatyushaX^s and BSGD-IM-KatyushaX^s are applied in the above 2D simulation case with different f and τ, as shown in Fig.4.11. In the figure, the constant step lengths in BSGD(-IM) are well tuned to ensure the fastest reconstruction speed, which are denoted by green lines. BSGD(-IM) with KatyushaX^s is presented by three different colors to denote different acceleration frequencies. Simulations show that with a $\tau < 0.5$ and a $f \geqslant 2M$, the BSGD-KatyushaX^s is able to accelerate the BSGD and the BSGD-IM-KatyushaX^s is able to accelerate the BSGD-IM. The $2M$ is experimentally showed to be an effective threshold value above which the KatyushaX^s begin to accelerate original BSGD(-IM).

4.9 Conclusions

BSGD can be viewed as an improvement of CSGD. It accumulates the previous sub-gradients, thus making the error variance of the estimated gradient decrease along with iterations. It has a faster reconstruction speed than CSGD, but the total storage amount of all variables has increased and the communication scheme is more complicated as well. It requires 2 inner-loops to update $\mathbf{r}$ and $\mathbf{x}$ separately. This communication is the same

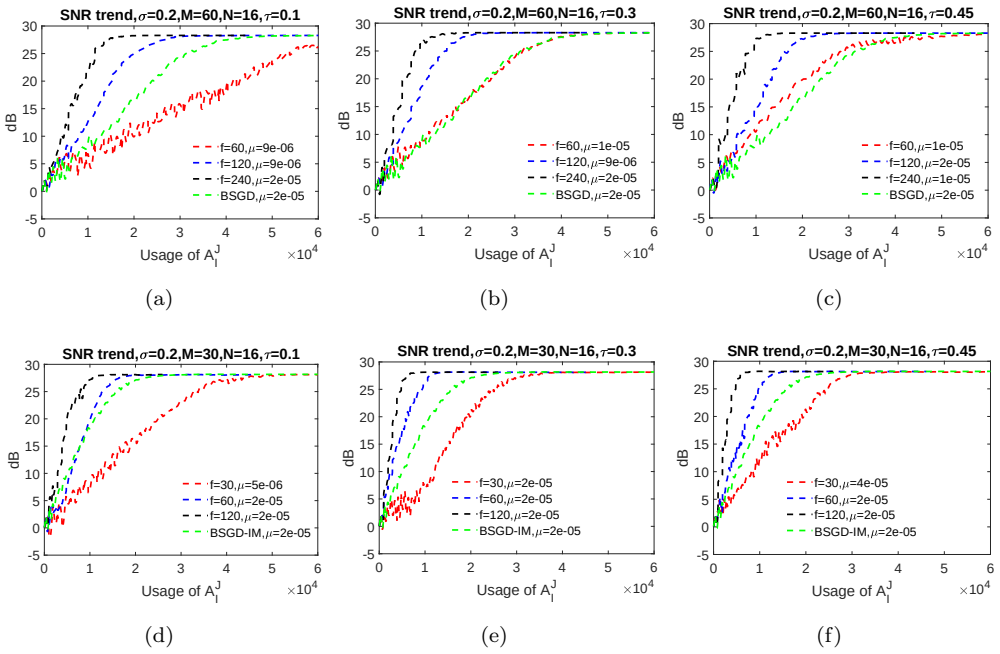


Figure 4.11: The comparisions of BSGD(-IM) with their KatyushaX^s acceleration trick. The first row is BSGD group and the second row is BSGD-IM group. BSGD(-IM) are denoted by green lines whilst the other three colours are accelerated references.

as the other SGD-type methods such as SAG and SVRG. However, BSGD does not have to iterate over all blocks of $\mathbf{x}$ each time since it is able to estimate a high accuracy solution even when $\gamma < 1$. It thus provides flexible sampling of sub-matrix $\mathbf{A}_I^J$. In this chapter, the sampling method is the same as that in CSGD. The data communication is minimized by setting $\alpha = \frac{1}{M}$ and adjusting γ according to the number of actual parallel computation nodes. This sampling method enables $\mathbf{x}$ blocks to be updated within the parallel computation nodes. Simulation results have shown that under noise free case, the BSGD can approach to the true solution. This is mainly reflected in the results shown from Fig.4.1 to Fig.4.3.

The importance sampling is also applicable to BSGD and simulations have verified that BSGD-IM has the fastest reconstruction speed compared with other existing SIRT-type methods including mini-batch SGD, SAG, SVRG. An automatic parameter tuning strategy is also proposed to accelerate the case when the initial step-length is set improperly small or large. In addition, KatyushaX^s acceleration is applied to BSGD. Instead of applying a momentum step after every single stochastic iteration, the BSGD-KatyushaX^s method applies a momentum acceleration with an update frequency f. Simulations show that when $f \geq 2M$, the BSGD-KatyushaX^s and BSGD-IM-KatyushaX^s effectively accelerate the BSGD and BSGD-IM algorithms, even when they are well tuned to the fastest reconstruction speeds.

The open area of this chapter is also the convergence property of BSGD. Similar to CSGD, the main research interest of BSGD focuses on the reconstruction speed of BSGD under different parameter settings and how to further accelerate the speed. As a result, the convergence analysis of BSGD is not fully illustrated by simulations. In this chapter, SNR trends, which can reach around 120 dB in the noise free model and single-precision floating-point format, is believed to be able to show that the BSGD is able to closely approach to the true solution. Besides, the DS trends shown in Fig.4.7, Fig.4.9 and Fig.4.10, which directly reflect the distance of the reconstructed image vector to the least square solution, shows that the BSGD approaches to the least square solution much closer than CSGD and achieves the same level with SAG and SVRG. As a result, the conjecture is that the BSGD converges on expectation to the least square solution. This conjecture is not fully proved. In fact, for both CSGD and BSGD, only the SNR trend at the initial stage is paid attention. This is because that the early-stopping strategy is adopted to avoid the semi-convergence property, which is not presented in this thesis because of the model is not severely ill-posed and the noise is not severe. The initial increasing SNR trend is also related to the decrease of the data fidelity, which hints that both CSGD and BSGD can be possible to be incorporated in the proximal method system. This will be discussed in the next chapter.

Chapter 5

Adding TV regularisation

In Chapter 3 and Chapter 4, two parallel algorithms have been proposed and compared with each other as well as with other SIRT-type methods. The algorithms empirically converged to least squares or weighted least squares solutions, which are only good estimates if enough data is available and if the system matrix is well conditioned. As the CT reconstruction problem is typically very ill-conditioned, this chapter shows how the efficient computational method of the previous sections can be adapted to include explicit regularisation. This chapter uses the TV regularisation term as an example regulariser.

To be more specific, This chapter discusses the case in which we have fewer measurements than unknowns, especially when we have few, but equally spaced projections covering a full scan circle. Under this case the linear system is severely ill-posed. As a result, a regularization terms is needed. As discussed in section 2.3, Tikhonov regularization and TV regularization is two popular regularizations. According to the difference discussed in section 2.3, TV regularization is more suitable for the case when the projection views are sparse or limited. As a result, in this chapter the regularization term is constrained as TV regularization.

We evaluate the reconstruction quality provided by BSGD/CSGD combined with TV denoising (i.e. BSGD/CSGD-TV) to approximately optimize the objective function containing TV regularizations. The combination is inspired by the proximal methods like ISTA and FISTA. In the FISTA, for example, solving the TV constrained objective function Eq.2.44 includes two steps: the first step is to reduce the data fidelity (i.e. reducing $\|\mathbf{y} - \mathbf{A}\mathbf{x}_{rec}\|^2$), followed by the second step, a TV de-noising procedure (Beck and Teboulle, 2009b). Many literature have revealed that in the fist step, not only the GD can be used to reduce the data fidelity, many SGD-type algorithms can also be used to substitute GD to reduce the iteration computation overhead. Those stochastic algorithms include SVRG, SGD and even delayed SGD where the stochastic gradient calculated by a row block of $\mathbf{y}$ is further blurred by an extra noise vector (Schmidt

et al., 2011; Beck and Teboulle, 2009b,a; Nitanda, 2014). Consider the fact that simulations have verified that the CSGD/BSGD can reduce the data-fidelity just like the other mature stochastic methods, the combination of TV denoising and CSGD/BSGD should also be feasible to approximately optimize the objective function Eq.2.44. Here the "approximately" hints that the CSGD-TV and BSGD-TV may not be able to minimise the TV-constrained objective function. However, simulations will show that the reconstructed result will have minor difference with result obtained from FISTA.

Generally speaking, CSGD/BSGD is used to reduce the data fidelity as the first step, then a TV de-noising procedure is performed. The algorithm is shown in Algo.5.1, where

Algorithm 5.1 BSGD/CSGD-TV

1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\mathbf{g} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]} = \mathbf{0}$, and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
2: **for** $epoch = 1, 2, \cdots, K_{max}$ **do**
3: **for** t=1,2,$\cdots$,f **do**
4: [**x**,**z**,**g**]=BSGD^{1ep}(**x**,μ,α,γ,**z**,**g**)
5: or
6: [**x**,**z**,**r**]=CSGD^{1ep}(**x**,μ,α,γ,**z**,**r**)
7: **end for**
8: $\mathbf{x} = \arg\min_{\mathbf{u}} \|\mathbf{u} - \mathbf{x}\|^2 + 2\mu\lambda TV(\mathbf{u})$
9: **end for**
10: $\mathbf{x}_{solution} = \mathbf{x}$

BSGD^{1ep} has been defined in Algo.4.7. CSGD^{1ep} has a similar definition, performing lines 3-15 in Algo.3.3. According to the discussions in Fig.4.5 and Fig.4.6, in a master-servant parallel network, all variables $(\mathbf{x}, \mathbf{r}, \mathbf{g}, \cdots)$ are stored in the master node whilst the parallel computation nodes do not have full access to them. As a result, the TV de-noising procedure (line 8 in Algo.5.1), which is performed on all of $\mathbf{x}$, is initially performed in the master node as discussed in section 5.1. Simulations show the effectiveness of BSGD/CSGD-TV by comparing the SNR trend of $\mathbf{x}$ and the reconstruction results with other algorithms. Since the TV norm is a sum of l_2 norms of the pixel gradients, it has the potential to be parallized over the computation nodes. As a result, in section 5.2, the TV de-noising procedure is moved from the master node to parallel computation nodes to further parallelize the BSGD/CSGD-TV. To reduce the communication cost between master node and computation nodes, the TV de-noising is applied on partial image blocks $\mathbf{x}_J$ instead of the whole set $\{\mathbf{x}_{J_j}\}_{j=1}^N$. Simulations prove the effectiveness of such combinations of BSGD/CSGD and TV regularizations, suggesting that BSGD/CSGD-TV have a parallel computation scheme with more flexibility and less communication overhead than other SIRT-type algorithms.

5.1 TV de-noising on the whole of x

In this section, CSGD/BSGD-TV are compared with the following popular methods: GD-TV(ISTA), Fast Iterative Shrinkage-Thresholding Algorithm(FISTA) (Beck and Teboulle, 2009b), SVRG-TV,SAG-TV. To compare reconstruction speeds, it is important to unify the frequency of applying TV de-noising. For example, assume that $\mathbf{y}, \mathbf{x}$ space are divided into M, N blocks for all methods. ISTA/FISTA then performs a TV de-noising step after $2MN$ matrix-vector multiplications while SAG and SVRG, which generally calculate a stochastic gradient using a row block per iteration, perform the TV de-noising step after $2N$ matrix-vector multiplications. In this case, it is not appropriate to compare each other's reconstruction speed using "the usage of $\mathbf{A}_I^J$" due to the different number of TV de-noising steps used. To ensure the "usage of $\mathbf{A}_I^J$" measure is still proportional to computing time, in this section, the application frequency of TV de-noising is unified: TV de-noising is performed only after performing $2MN$ matrix-vector multiplications. As a result, f in CSGD-TV is $\lceil \frac{2}{3\alpha\gamma} \rceil$ and in BSGD-TV is $\lceil \frac{1}{\alpha\gamma} \rceil$. Besides, the iteration number within TV de-noising procedure is also unified as 20 to unify the computation amount.

An ASTRA-generated 2D scanning geometry with few projections is adopted. The scanning geometry is shown in Table.5.1, the default pixel size for image and CCD size for detector are both 1. The definitions of geometry parameters are mentioned in Fig.3.5.

Table 5.1: 2D scanning geometry for few-views projections

K	Detector size	OP	OD
256	550	300	300
Projection view gap	Projection view number	Noise variance	SNR of y
10	36	1	31.0dB

Using this geometry, the simulations results are shown in Fig.5.1. In the figure, the step-length μ and the TV regularization parameter λ are carefully selected from a wide range to ensure the best performance (i.e. the highest SNR limit) of all algorithms. Especially for λ, it is selected from a wide range [0.1, 1, 10, 50, 100]. It can be seen that BSGD/CSGD-TV can both reconstruct the image into a high accuracy solution, despite that the final solution is different from each other, which means that they converge to different locations. However, these solutions' locations are near to the location of the actual image vector. BSGD-TV shows the fastest reconstruction speed among all candidate methods. CSGD-TV is fast at the initial stage, while the final location of iterated $\mathbf{x}$ stays a bit far from the location of $\mathbf{x}_{true}$, especially when compared with BSGD-TV and FISTA.

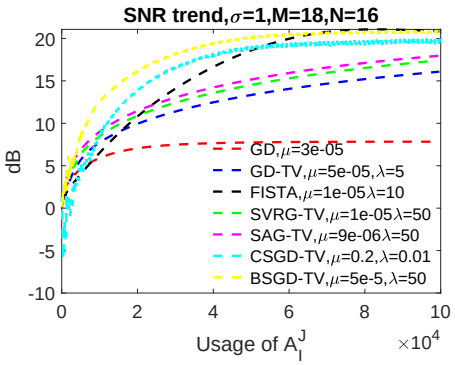


Figure 5.1: SNR trends for different methods to reconstruct the $\mathbf{x}_{rec}$. For CSGD/BSGD, $\alpha = \frac{1}{18}, \gamma = \frac{4}{16}$.

To prove the effectiveness of CSGD/BSGD-TV, the reconstruction results after different iterations are shown in Fig.5.2. It can be seen that the initial reconstructed images obtained from CSGD/BSGD-TV inevitably exist artefacts around the image block margins, but these artefacts gradually disappear as iterations go on. The final images provided by CSGD/BSGD-TV do not have significant difference from the other existing first-order methods but enjoys a reduced communication overhead in the reduction of data fidelity, especially when p is smaller than image block number N, as discussed in Chapter 4.

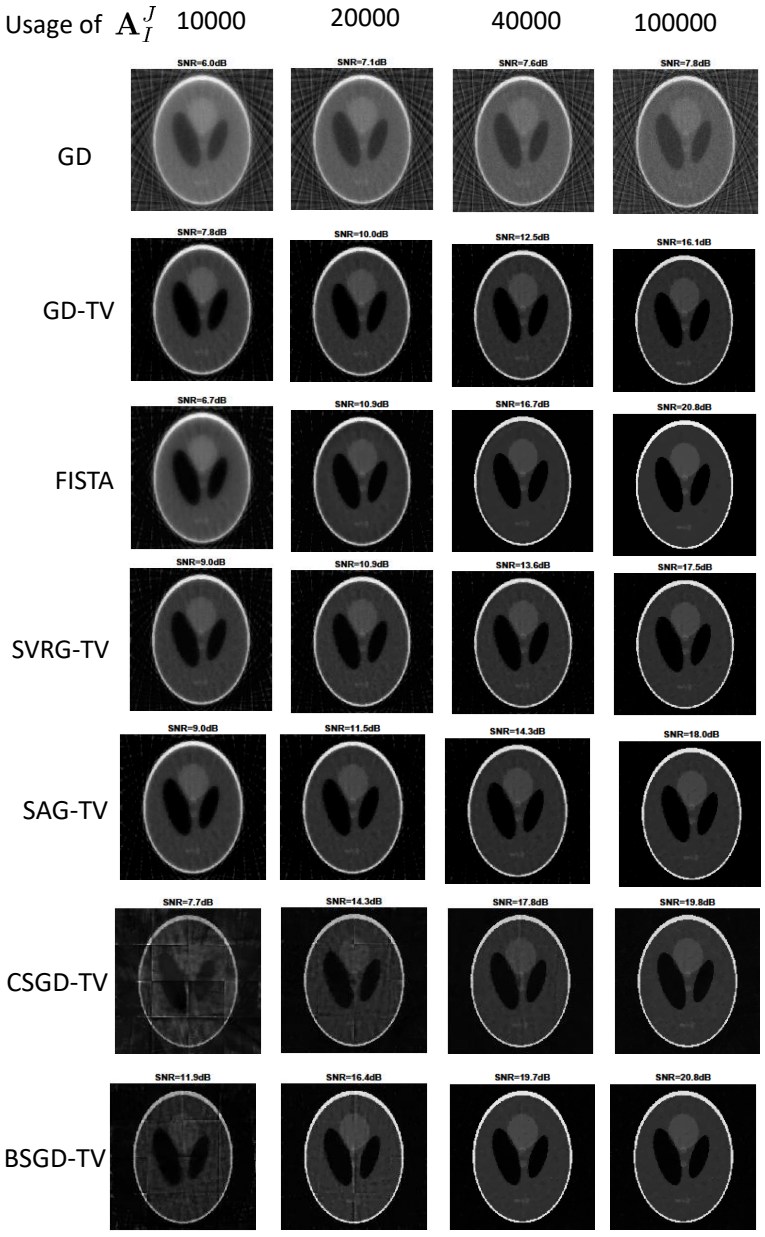


Figure 5.2: Reconstruction results after different projection times of different methods.

5.2 TV de-noising on parts of $\mathbf{x}_J$

In the above section, the TV constraints are applied on the whole of $\mathbf{x}$. This process can be viewed as performed by the master node since parallel nodes do not have full access to $\mathbf{x}$. In this section, the TV constraints are moved to parallel computation nodes and are applied on $\mathbf{x}_J$ separately. There are three reasons why this research is of interest here: 1) When $\mathbf{x}$ is large, it is possible that $\mathbf{x}$ is stored in a distributed way and thus it is separated into different files. As a result, even the master node cannot have easy access to the whole of $\mathbf{x}$. 2) The master node may lack calculation ability and only has storage ability. For example, the master node is one or several hard discs

and the computation nodes are several GPU/CPUs. In this case, the hard discs have to transport the whole of $\mathbf{x}$ to one node when it needs to be de-noised, which increases the communication overhead. 3) The computation complexity of TV de-noising methods increases rapidly when the size of image increases. Applying TV de-noising on the whole of $\mathbf{x}$, due to the lack of parallel computation ability, can be much slower than separately and simultaneously de-noise $\{\mathbf{x}_{J_j}\}_{j=1}^{N}$. To verify this assumption, the TV de-noising algorithm is applied on different size of images and the time spent on each case is shown in Fig.5.3.

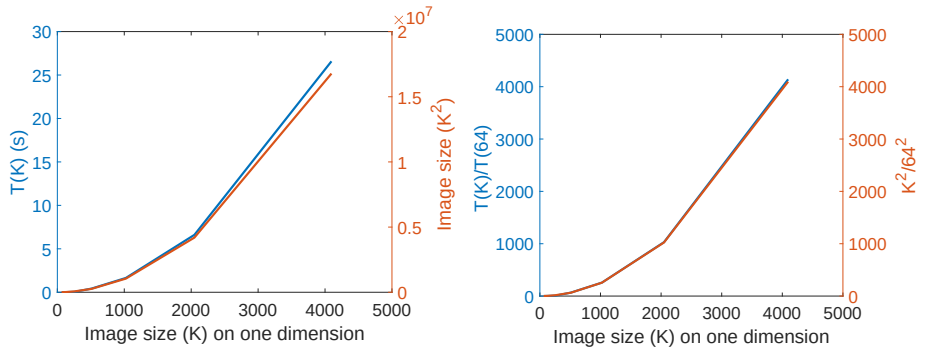


Figure 5.3: The iteration number of TV de-noising is predetermined as 20. The image size K is $[64, 128, 256, 512, 1024, 2048, 4096]$. (a) shows that the time spent on TV de-noising ($T(K)$) for different image sizes K. (b) shows that the increase speed of $T(K)$ is almost the same with that of the image size. This suggests that if $\mathbf{x}$ is divided into N sub-blocks and are de-noised among N parallel nodes, the total time spent on TV de-noising can be N times shorter than non-splitting case if the communication cost is ignored.

One simple method to parallelize the TV part and to reduce the computation cost is to approximate the TV term as a sum over $\mathbf{x}$ space, i.e.

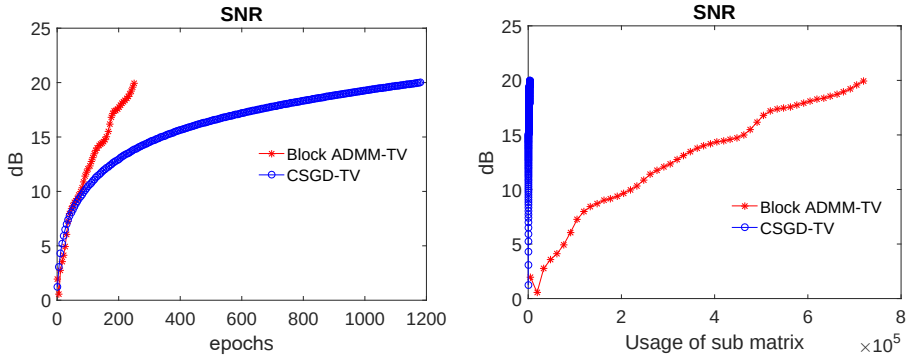
$$TV(\mathbf{x}) \approx \sum_{j=1}^{N} TV(\mathbf{x}_{J_j}). \tag{5.1}$$

This approximation enables the block ADMM, which is mentioned in Chapter 3, to be a candidate parallel method to solve the approximated TV-constrained optimization problem:

$$\begin{aligned} &\arg\min \underbrace{\frac{1}{2}\sum_{i=1}^{M} \|\mathbf{y}_{I_i} - \mathbf{A}_{I_i}\mathbf{x}\|^2}_{f(\mathbf{y})} + \underbrace{\lambda \sum_{j=1}^{N} TV(\mathbf{x}_{J_j})}_{g(\mathbf{x})}, \\ &s.t. \mathbf{y} = \mathbf{A}\mathbf{x}, \end{aligned} \tag{5.2}$$

where $f(\mathbf{y})$ and $g(\mathbf{x})$ are both block separable functions, as mentioned in Eq.2.57. Detailed iteration methods have been discussed in session 2.4 and thus only simulation results are presented in this section. Block ADMM-TV is directly compared with CSGD-TV as proposed in Algo.5.1. To satisfy the constraint in Eq.5.2, σ in Table 5.1 is currently

set as 0. The parameter choice for block ADMM-TV is carefully selected as discussed in section3.2.2.1 to ensure the best reconstruction speed. The reconstruction speed of block ADMM-TV and CSGD-TV is shown in Fig.5.4. It can be seen that these two methods have similar initial reconstruction speeds. Although the reconstruction speeds of CSGD-TV becomes slow as iteration goes on, it uses significantly fewer matrix-vector multiplications to achieve the predefined SNR.



(a) (b)

Figure 5.4: $TV(x)$ is approximated as $\sum_{j=1}^{N} TV(\mathbf{x}_{J_j})$. Reconstruction speed between CSGD-TV and block ADMM-TV clearly shows that the CSGD-TV are much faster than ADMM-TV.

The images reconstructed by the two methods are compared and they show different reconstruction quality, as shown in Fig.5.5.

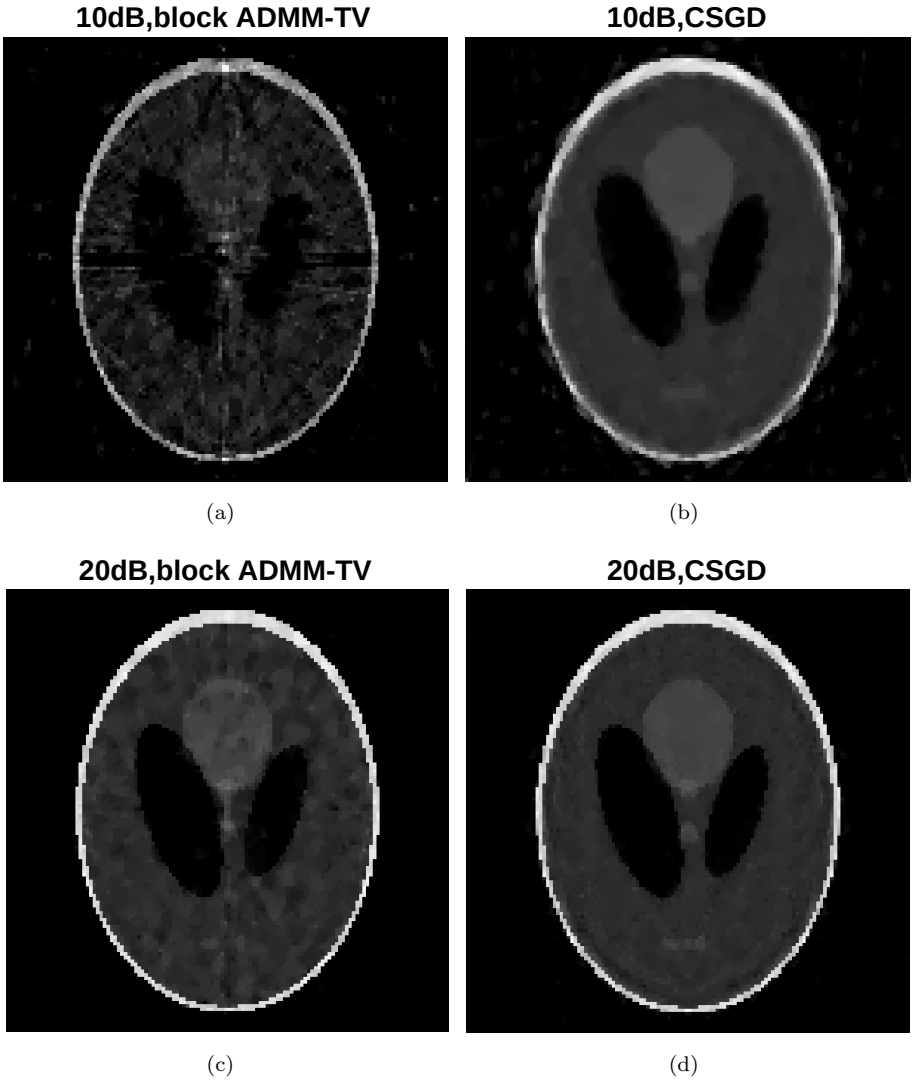


Figure 5.5: Reconstructed images from ADMM-TV and CSGD-TV. The ADMM-TV exists more severe inner artifacts than CSGD-TV, suggesting that directly approximate $TV(\mathbf{x})$ as a sum-up over $\mathbf{x}$ space is not an appropriate choice for ADMM-TV.

It can be seen that the CSGD-TV method presents visually smoother images whilst severe inner noise and artefacts at inner margins exist in block ADMM images. As a result, in CT reconstruction, dividing the TV term according to the partition on $\mathbf{x}$ space and then applying the block ADMM method is not appropriate. This is because that strictly speaking, the TV constraint defined in Eq.2.43, is not separable since the calculation always ignores the overlapping margin (x and y in Eq.2.43 often begin with 2 rather than 1). If the image is divided into many sub-images, the real TV norm of the original intact image is not a simple accumulation of TV norms of each sub-image blocks i.e. $TV(\mathbf{x}) \neq \sum_{j=1}^{N} TV(\mathbf{x}_{J_j})$. If $TV(\mathbf{x}_{J_j})$ is minimised separately, the border generated by $\mathbf{x}_{J_j}$ cannot be effectively de-noised, which causes artefacts at the margins among neighbouring sub-image blocks, leading to $\arg\min_{\mathbf{x}} \|TV(\mathbf{x})\| \neq \cup_{j=1}^{N} \arg\min_{\mathbf{x}_{J_j}} \|TV(\mathbf{x}_{J_j})\|$.

A phantom image is used to verify the inaccuracy of minimizing TV norms in blocks. The phantom image is firstly de-noised without splitting (i.e. minimizing $TV(\mathbf{x})$) as a "reference solution", as shown in Fig.5.6. The phantom image is then divided into N

Figure 5.6: The TV de-noising result of intact phantom image, which is used as a reference solution when $\mathbf{x}$ is divided into N blocks and the TV de-noising is performed on each $\mathbf{x}_J$. The number of iterations in the TV de-noising process is predefined as 20.

parts, as illustrated in Fig.5.7. Those divided sub-images are de-noised in parallel and

Figure 5.7: Partition the image into $N = 4$ blocks. If each block get de-noised separately, then the inner margins will be ignored due to the definitions of TV norm, thus affecting the approximation of the TV de-noising on the whole image. "inner margins" here refer to the boundaries between neighbouring sub-image blocks. For example, the inner margins of image block $\mathbf{x}_{J_1}$ are the right and bottom margins.

the results are assembled. The comparisons of the assembled solution and reference solution are shown in Fig.5.8. Simulation results clearly show that $TV(\mathbf{x})$ term cannot be directly divided in $\mathbf{x}$ space, which explains why the block ADMM-TV performs poorly in this case.

To overcome the influence brought by inner margins, when de-noising $TV(\mathbf{x}_J)$, $\mathbf{x}_J$ need to "borrow" extra pixels, with width w pixels, from its neighbouring image blocks

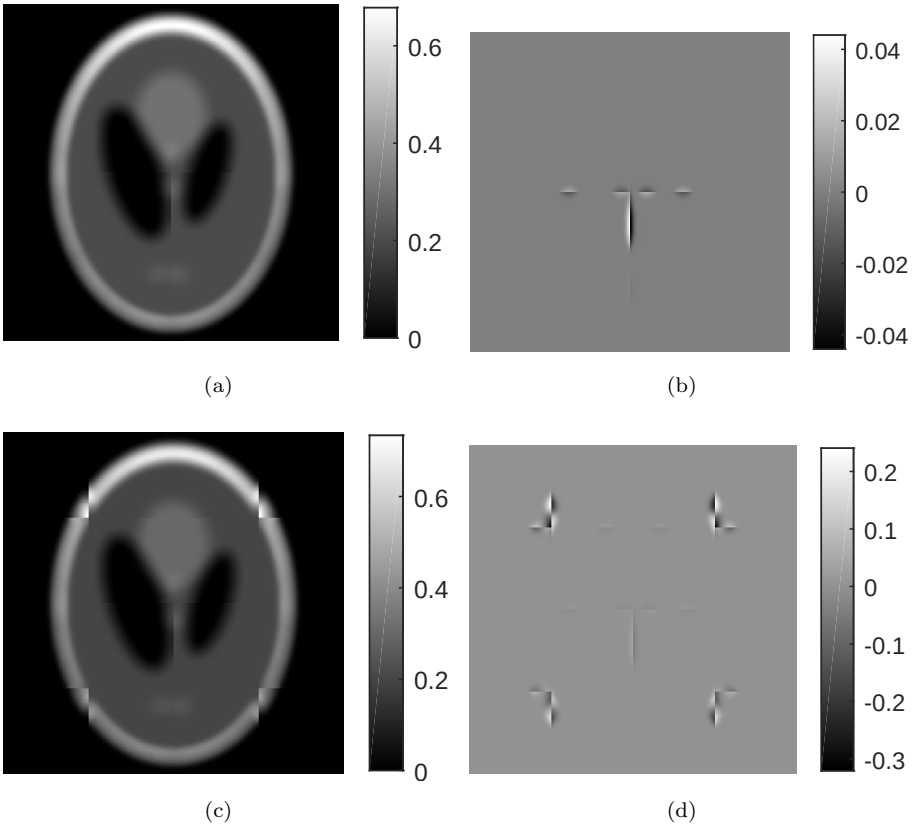
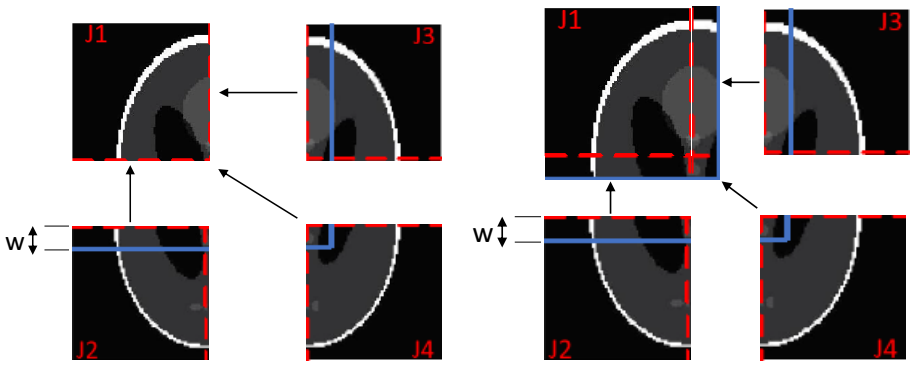


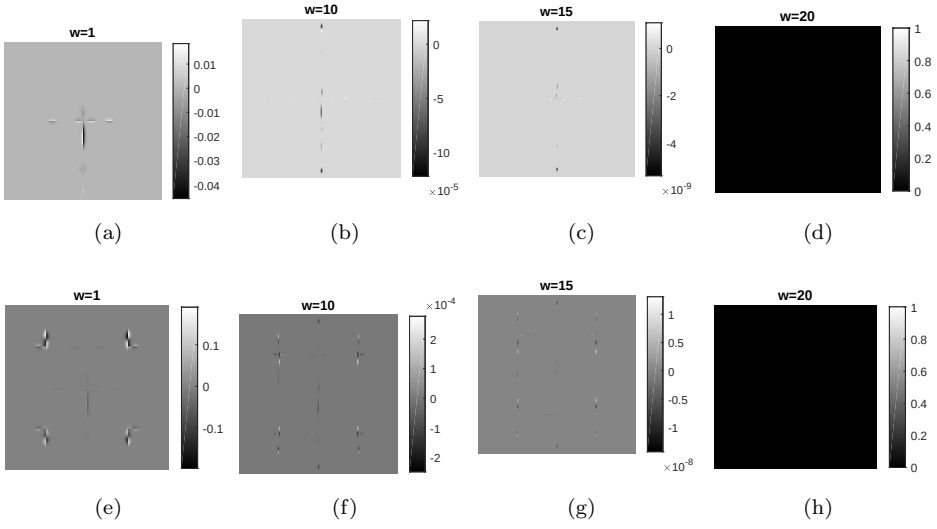
Figure 5.8: TV de-noising is applied on separate sub-images and the iteration number is also 20, which is the same as Fig.5.6. (a) is the assembled TV de-noising result when halving each image dimension($N = 4$). (b) is the difference between assembled solution and reference solution shown as Fig.5.6. It clearly shows that at the "inner margins" which are generated by sub-image blocks, the difference are significant. (c) and (d) are repeated simulation results when dividing each image dimension into 4 parts ($N = 16$). The differences still mainly exist around the inner margins.

to wrap its "inner margins". For $\mathbf{x}_{J_1}$ in Fig.5.7, its neighbouring image blocks are $\{\mathbf{x}_{J_j}\}_{j=2,3,4}$. The borrowing process is shown in Fig.5.9. When w is 1,10,15,20, the differences between reference solution and assembled solution are shown in Fig.5.10.



(a) (b)

Figure 5.9: (a) $\mathbf{x}_{J_1}$ borrows some extra pixels, with width w pixels, from its neighbouring image blocks. (b) The new image block is named as $\tilde{\mathbf{x}}_{J_1}$. When TV de-noising procedure is applied after date fidelity reduction, the target image block should be $\tilde{\mathbf{x}}_{J_1}$ instead of $\mathbf{x}_{J_1}$. After the de-noising process, the borrowed pixels should be deleted. The above procedure is applied in parallel for all $\{\mathbf{x}_{J_j}\}_{j=1}^{N}$.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.10: The first row shows that when N is 4, the differences between assembled solution and reference solution gradually disappear when w increases to 20. The second row is the repeated simulation when N is 16. As a result, for different partitions, setting $w = 20$ makes the separate TV de-noising have the same results with non-partition case.

To further verify the influence of w on the accuracy of separate TV de-noising, different image sizes and division methods are compared. The relative error RE here is defined as $\frac{\|\mathbf{x}_{seperate}-\mathbf{x}_{whole}\|}{\|logic(\mathbf{x}_{seperate}-\mathbf{x}_{whole})\circ\mathbf{x}_{whole}\|}$, where $\mathbf{x}_{seperate}$ is the result obtained by applying TV de-noising on separate $\{\mathbf{x}_{J_j}\}_{j=1}^{N}$ and $\mathbf{x}_{whole}$ is one obtained by applying TV on $\mathbf{x}$ without division. The $logic(\mathbf{v})$ function sets all non-zero elements in $\mathbf{v}$ as 1. The "$\circ$" is the Hadamard product operation. The simulation results are shown in Table 5.2. It can be

Table 5.2: RE changing trend under different image size K and partition number$N$

K	N	w=1	w=10	w=15	w=20
256	4	1.90E-02	4.80E-05	2.40E-09	0
256	16	8.30E-02	9.20E-05	7.00E-09	0
512	4	6.30E-03	2.30E-05	1.30E-09	0
512	16	7.30E-02	6.40E-05	4.10E-09	0
1024	4	4.30E-03	1.70E-05	1.80E-09	0
1024	16	5.10E-02	4.00E-05	2.60E-09	0

seen that when $w = 20$, the separate TV de-noising is the same as the whole TV de-noising regardless the image size and partition methods. As a result, the TV de-noising process can be moved to the parallel computation nodes and the algorithm is shown in Algo.5.2. f in the Algo.5.2 should be the same as in Algo.5.1. This means that for

Algorithm 5.2 BSGD/CSGD-TV, applying TV on $\{\mathbf{x}_{J_j}\}_{j=1}^{N}$

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$,$\mathbf{g} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]} = \mathbf{0}$, and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
for *epoch* $= 1, 2, \cdots, K_{max}$ **do**
  for t=1,2,$\cdots$,f **do**
    [$\mathbf{x}$,$\mathbf{z}$,$\mathbf{g}$]=BSGD^{1ep}($\mathbf{x}$,μ,α,γ,$\mathbf{z}$,$\mathbf{g}$)
    or
    [$\mathbf{x}$,$\mathbf{z}$,$\mathbf{r}$]=CSGD^{1ep}($\mathbf{x}$,μ,α,γ,$\mathbf{z}$,$\mathbf{r}$)
  end for
  for j=1,2,$\cdots$,N in parallel **do**
    "Wrap" $\mathbf{x}_{J_j}$ to $\tilde{\mathbf{x}}_{J_j}$ with width $w = 20$
    $\tilde{\mathbf{x}}_{J_j} = \arg\min_{\mathbf{u}} \|\mathbf{u} - \tilde{\mathbf{x}}_{J_j}\|^2 + 2\mu\lambda TV(\mathbf{u})$
    "Dewrap" $\tilde{\mathbf{x}}_{J_j}$ to $\mathbf{x}_{J_j}$ with width $w = 20$
  end for
end for
$\mathbf{x}_{solution} = \mathbf{x}$

each epoch, the computation amount includes: $2MN$ matrix-vector multiplications + N partial TV de-noising operations.

Assume that p is smaller than N and can divide N without remainder, it is clear that each parallel node needs to communicate with the master node $\frac{N}{p} = \frac{1}{\gamma}$ times. Thus it is natural to consider only performing TV de-noising on p image blocks instead of on all blocks to reduce the communication times between computation nodes and the master node. However, if f remained unchanged, this means that each iteration contains: $2MN$ matrix-vector multiplications + γN partial TV de-noising operations, thus making the "the usage of $\mathbf{A}_I^J$" not an appropriate time reference in this section. As a result, when

the TV de-noising is only performed on γN image blocks, then the original f should also be γ times smaller than before. The algorithm is shown in Algo.5.3, where f for CSGD

Algorithm 5.3 BSGD/CSGD-TV, applying TV on γN $\{\mathbf{x}_{J_j}\}_{j\in[1,N]}$

1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\mathbf{g} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]} = \mathbf{0}$, and $\mathbf{r} = \mathbf{y}$. α and γ are the percentage of the selected row and column blocks respectively.
2: **for** *epoch* $= 1, 2, \cdots, K_{max}$ **do**
3: **for** t=1,2,$\cdots$,f **do**
4: [**x**,**z**,**g**]=BSGD^{1ep}(**x**,μ,α,γ,**z**,**g**)
5: or
6: [**x**,**z**,**r**]=CSGD^{1ep}(**x**,μ,α,γ,**z**,**r**)
7: **end for**
8: Computation nodes again request latest $\mathbf{x}_{J_j}$ from the master node in parallel
9: **for** the selected γN $\mathbf{x}_{J_j}$ in parallel **do**
10: "Wrap" $\mathbf{x}_{J_j}$ to $\tilde{\mathbf{x}}_{J_j}$ with width $w = 20$
11: $\tilde{\mathbf{x}}_{J_j} = \arg\min_{\mathbf{u}} \|\mathbf{u} - \tilde{\mathbf{x}}_{J_j}\|^2 + 2\mu\lambda TV(\mathbf{u})$
12: "Dewrap" $\tilde{\mathbf{x}}_{J_j}$ to $\mathbf{x}_{J_j}$ with width $w = 20$
13: **end for**
14: **end for**
15: $\mathbf{x}_{solution} = \mathbf{x}$

is decreased from $\lceil \frac{2}{3\alpha\gamma} \rceil$ to $\lceil \frac{2}{3\alpha} \rceil$ and for BSGD is decreased from $\lceil \frac{1}{\alpha\gamma} \rceil$ to $\lceil \frac{1}{\alpha} \rceil$. The SNR trend and reconstructed images when using Algo.5.3 are shown in Fig.5.11 and Fig.5.12.

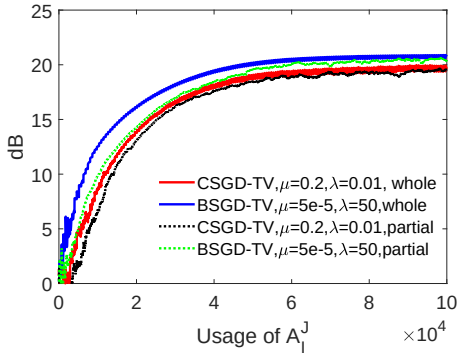


Figure 5.11: "whole" in the figure means that the TV de-noising is applied on the whole of $\mathbf{x}$, while the "partial" means that the TV de-noising is applied on γN selected image blocks $\mathbf{x}_J$. The partial TV de-noising has some negative effects on final solutions' accuracy, especially making the final solution of CSGD-TV have a lower solution accuracy, but in general the trend is very similar to the original case.

Simulations have verified that Algo.5.3 can effectively reconstruct $\mathbf{x}$ to the same accuracy level as Algo.5.1, but with advantages including: 1) the ability to incorporate the TV de-noising into parallel computation nodes, and 2) no further significant communication

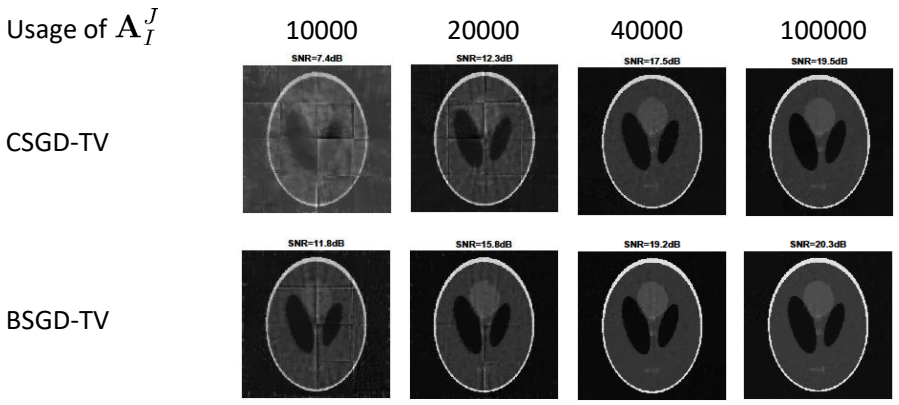


Figure 5.12: The reconstructed images of BSGD/CSGD-TV(partial). The are similar to the previous results provided by BSGD/CSGD-TV(whole), especially after 40,000 usage of $\mathbf{A}_I^J$.

overheads are required between master node and computation nodes. The "Wrap" process indeed requires some extra communication overhead but since the width w only needs to be 20 pixels for all problem sizes, thus the extra data communication can be ignored especially when $\mathbf{x}$ is huge.

5.3 Conclusions

In this chapter the combinations of BSGD/CSGD and TV regularization is experimentally shown to be effective. When the projection data is limited, then TV regularization leads to better reconstruction quality. However, the introduced TV de-noising process can be time-consuming, especially when the image size is huge. In traditional GD-TV/FISTA, the TV de-noising process is performed after a full gradient calculation, which requires $2MN$ usage of $\mathbf{A}_I^J$, whiles in the other TV-constrained optimization algorithm, such as SVRG-TV, SAG-TV, the TV de-noising is performed after a stochastic gradient calculation, which requires $2N$ usage of $\mathbf{A}_I^J$. The difference in the frequency with which TV de-noising is performed makes the "usage of $\mathbf{A}_I^J$" no longer reflect the actual wall time during reconstruction. To overcome this issue, in this chapter the frequency of performing TV de-noising in all methods is unified, including SVRG-TV,SAG-TV and the new proposed CSGD/BSGD-TV. Only after $2MN$ usage of $\mathbf{A}_I^J$ then the TV de-noising is performed. With this unified de-noising frequency, BSGD-TV shows the fastest reconstruction speed, achieving a high-SNR solution within the least usages of $\mathbf{A}_I^J$. Apart from the speed advantage, BSGD-TV, as well as CSGD-TV, have another advantage, which is the ability to perform TV de-noising in a parallel and stochastic way. The TV norm, according to the definition, is a sum of l_2 gradient norms. However, it cannot address pixels located at margins. As a result, directly splitting image $\mathbf{x}$ into N blocks and then performing TV de-noising on all partial image block $\mathbf{x}_J$ does not strictly equal to the case when performing TV de-noising on the whole of $\mathbf{x}$. A "wrap" scheme

is proposed. When performing a TV de-noising on $\mathbf{x}_J$, this allows $\mathbf{x}_J$ to temporarily borrow some pixels with width w from its neighbouring image blocks, thus the inner margins of $\mathbf{x}_J$ are wrapped by new borrowed pixels. Similarly, when the TV de-noising is finished, a "de-wrap" procedure is applied to remove those borrowed pixels. Simulations have verified that when w is 20 pixels, $\arg\min_{\mathbf{x}} \|TV(\mathbf{x})\| = \cup_{j=1}^{N} \arg\min_{\mathbf{x}_{J_j}} \|TV(\mathbf{x}_{J_j})\|$. This property enables the TV de-noising to be performed within parallel computation nodes, where each of them has limited access to the whole of $\mathbf{x}$. When p is smaller than N, $\arg\min_{\mathbf{x}} \|TV(\mathbf{x})\|$ then requires the communications between the master node and computation nodes to be larger than one time. To reduce the communication cost, the TV de-noising procedure adopts a similar partial update strategy as BSGD/CSGD. It does not cover all $\{\mathbf{x}_{J_j}\}_{j=1}^{N}$. Instead, it only addresses γN image blocks, where γ follows the proposed selection criteria shown in Eq.3.8. Simulations have verified that this partial update strategy, named as BSGD/CSGD-TV(partial), reconstructs the scanned object into high-SNR solution without sacrificing speed compared with BSGD/CSGD-TV(whole), which performs TV de-noising on the whole of $\mathbf{x}$.

Whilst we have here looked at TV constraints, other regularisers could be treated in a similar way. However, this was not explored here.

Chapter 6

Application on parallel architectures

The previous chapters have developed two algorithms and studied their basic mathematical properties, including reconstruction speed comparisons and their ability to be combined with TV regularizations. In this chapter, the research no longer focuses on the algorithms' mathematical properties but instead focuses on the realistic application of the algorithms in concrete parallel computing architectures. The two proposed algorithms show different performance in different scenarios. This is because the two algorithms use different communication schemes. This chapter will discuss this phenomenon in detail. Furthermore, in realistic applications, an asynchronous form of BSGD is proposed to further accelerate the reconstruction speed in certain settings.

The contributions of the chapter includes: 1) The CSGD and BSGD have been first applied in a realistic parallel network to measure their scalability. The measurement to reflect the reconstruction speed is changed from the "usage of $\mathbf{A}_I^J$" to the realistic wall time. Using this measurement, experiments have proved that the BSGD outperforms CSGD in terms of reconstruction speed. 2) An asynchronous BSGD is first proposed, enabling the fastest node to perform more computations and do not have to wait for the slowest node. This asynchronous communication has increased the communication efficiency and experiments have verified that it can outperform original BSGD algorithm if one node is significantly slower than the other nodes.

This chapter includes three parts. In section 6.1, the basic concepts and functions of parallel computation and communication, including point-to-point communication and collective communication, are introduced. Using these functions, the performances of BSGD/CSGD using different numbers of parallel computation nodes are presented and compared with each other in section 6.2 and 6.3. In the last section 6.4, the asynchronous communication scheme of BSGD is studied to illustrate that the reconstruction speed can be further increased when the parallel nodes have different computational abilities.

6.1 Basic concepts in parallel applications of BSGD/C-SGD

When applying BSGD/CSGD in a parallel network, it is required that each node communicates with each other node, sending data blocks such as $\mathbf{z}_I^j$,$\mathbf{x}_J$ etc. This kind of communication should follow the Message Passing Interface (MPI) standard. MPI is a message-passing standard that works on a wide range of parallel computing architectures. It provides a communication protocol for programming parallel computers and remains the dominant model used in high-performance computing today. MPI defines the syntax and semantics of a core of library routines that is useful to a wide range of users writing portable message-passing programs in many programming languages including C, C++, Fortran and Python (Barker, 2015).

In this chapter, the Python language together with the mpi4py package is used to implement different aspects of BSGD/CSGD when they are applied in a network with several computation nodes. In recent years, Python has occupied an increasing share in the field of numerical computing, and in the field of high-performance computing as well. This is because Python has become more than just a separate computer programming language, but has become a complete ecosystem composed of huge libraries and tools, as well as numerical calculations, including Numpy, Scipy, Pandas, etc. These numerical libraries and tools generally encapsulate and call efficient algorithm libraries implemented by Fortran, C, C ++, etc., so they make up for the shortcomings of Python's performance to a certain extent without compromising its flexibility and ease (Ceder, 2010; Bressert, 2012). In terms of the application of Python to high-performance parallel computing, there is a Python library built on top of MPI, mpi4py, which allows Python's data structures to be easily passed to multiple processes. Mpi4py is a very powerful library that implements many interfaces in the MPI standard, including point-to-point communication, collective communication, blocking / non-blocking communication, and inter-group communication. It also has good support for Python objects such as numpy arrays and they are very efficient to be communicated (Zaccone, 2015).

In mpi4py, some basic concepts will be used in future simulations. They are recalled below (Zaccone, 2015):

- process

  A "process" is an execution of a particular subroutine. In this thesis, one process will control one parallel computation node to perform the FP,BP or the updates on $\mathbf{r}, \mathbf{g}, \mathbf{x}$. In the following, different processes are distinguished by their $rank_i (i = 0, 1, \cdots, p-1)$.

- master node

The master node is used to store all necessary variables and send/receive data blocks to/from computation nodes. Unless specifically stated, it does not perform any projection operation or update on $\mathbf{r}, \mathbf{g}, \mathbf{x}$. In this thesis's illustrations, the master node is presented by a rectangle.

- computation node

  Computation nodes are mainly used to perform the projections. They request data blocks from the master node and then send results back to the master node. In this thesis's illustrations, the computation nodes are presented by ovals.

- root node

  The root node is also a computation node. It is responsible for the data collection and distribution when the collective communications are performed. It also performs the updates on $\mathbf{r}, \mathbf{g}, \mathbf{x}$. The definition of collective communications will be introduced below. In this thesis's figures, the root node is labelled as a blue oval to differentiate it from the other computation nodes (black oval).

- point-to-point communication

  Sending and receiving data are the two foundational concepts in mpi4py. Almost every single communication can be implemented with basic send and receive calls. The most general command to send and receive matrix data in mpi4py is "Send()" and "Recv()". They are block communications, which means that both functions will wait for each other until both of them have finished and then return back to each process for the following code to be executed. An illustration of "Send" and "Recv" is shown in Fig.6.1.

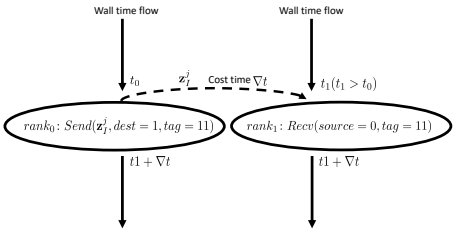


Figure 6.1: In a 2-process parallel network, $rank_0$ sends a $\mathbf{z}_I^j$ to $rank_1$. This process costs ∇t seconds. If $rank_0$ performs the *Send* command at time t_0 but $rank_1$ performs the *Recv* command at t_1 ($t_1 > t_0$), due to the block communication definition, $rank_0$ will "wait" between t_0 and t_1 and both $rank_0$ and $rank_1$ will finish data communication at the time $t_1 + \nabla t$. The "tag" in command is used to give a special name for the transferred data to ensure the receiver to receive the correct data.

During the block communication, the sender or receiver is not able to perform any other actions until the corresponding message has been sent or received. Blocking communications have a number of disadvantages. Potential computational time is simply wasted while waiting for the call to complete the block communication.

An alternative approach is to allow the program to continue execution while the messages is being sent or received. This is known as non-blocking communication. In mpi4py, non-blocking communication is achieved using the "Isend" and "Irecv" methods. The "Isend" and "Irecv" methods initiate a send and receive operation respectively, and then return immediately. These methods return an instance of the "Request" class, which uniquely identifies the started operation. The completion can then be managed by applying the "Test", "Wait", and "Cancel" functions to the Request class. An illustration of non-block communication is shown in Fig.6.2.

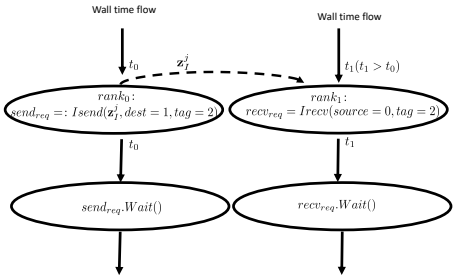


Figure 6.2: In the non-block communication case, each process does not wait for each other during the data communication. Instead, the "Isend" or "Irecv" returns immediately with a request class "$send_{req}$" or "$recv_{req}$". These requests will be checked to see whether the data communication is accomplished in the future using wait function.

- collective communications

  Collective functions involve communication among all processes in a process group (which means the entire process pool or a program-defined subset). A typical function is "Bcast". It is used to broadcast data from the root node to all processing units. An illustration of "Bcast" is shown in Fig.6.3.

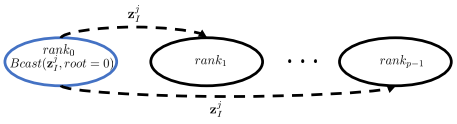


Figure 6.3: The "Bcast" is only performed on the root node $rank_0$. In this case, the root node will send $\mathbf{z}_I^j$ to all other parallel computation nodes.

"Scatter" is another data distribution function. It differs from broadcast, in that it does not send the same message to all processing units. Instead it splits the message and delivers one part of it to each processing unit. An illustration showing a matrix that is scattered to different nodes is shown in Fig.6.4.

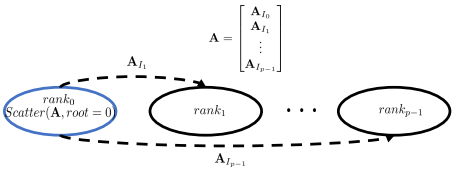


Figure 6.4: In this case, the "Scatter" function cuts the matrix into p row blocks and send them to $rank_0, \cdots, rank_{p-1}$ respectively.

"Gather" is a reverse function of "Scatter" function. It is used to store data from all processing nodes on a single processing nodes. An illustration is shown in Fig.6.5.

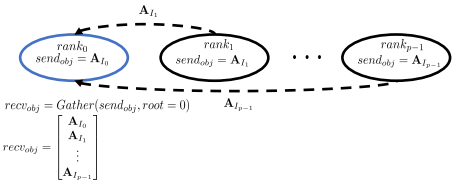


Figure 6.5: The "Gather" function is a reverse of "Scatter". Followed by Fig.6.4, this function collects row block $\mathbf{A}_I$ from all nodes and forms a new variable whose content is the intact $\mathbf{A}$.

"Reduce" function is used to collect data or partial results from different processing nodes and to combine them into one result by a chosen operator. Reduction can be seen as an inverse version of broadcast and the common operator includes "sum","min" and "max". An illustration using the "sum" operator is shown in Fig.6.6.

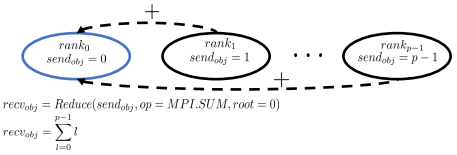


Figure 6.6: The "Reduce" function with sum operator sums up all $send_{obj}$ and stores in a new variable $recv_{obj}$ in $rank_0$.

These functions will be used in the next 2 sections to explore the scalability of two algorithms and the asynchronous BSGD form.

6.2 Synchronous application of BSGD/CSGD in CPU nodes

In previous BSGD or CSGD applications, each epoch uses $\alpha\gamma MN$ sub-matrices in one iteration and the number of parallel nodes (defined as p) is always assumed to be $\alpha\gamma MN$. Each node thus only processes one sub-matrix in each iteration. In this chapter this assumption no longer holds. This means that M, N, α, γ are pre-determined, and the actual number of parallel nodes (p) can be smaller than $\alpha\gamma MN$ and thus each node has to process more than one sub-matrix within one epoch, making the data communication more complicated. The speed-up under different p situations is measured. Here the speed-up for n nodes is defined as:

$$\text{speed-up} = \frac{\text{wall time using 1 nodes}}{\text{wall time using n nodes}} \tag{6.1}$$

If α, γ follow Eq.3.8, and α is set to $\frac{1}{M}$, which means that all parallel computation nodes always address the same row block I, then we can see that the update of $\mathbf{r}$ in both BSGD/CSGD is convenient. The update can be accomplished using "Reduce" which accumulates all $\nabla\mathbf{z}_I^j$ (the difference between updated $\mathbf{z}_I^j$ and previous one) from all parallel computation nodes. Furthermore, since $\alpha = \frac{1}{M}$, the image block $\mathbf{x}_J$ addressed by different computation nodes is also different from each other. As a result, when updating $\mathbf{x}$ in CSGD and $\mathbf{g}$ in BSGD, there is no data communication between computation nodes and the blocks of $\mathbf{x}, \mathbf{g}$ can be updated in each parallel computation node. These properties have already generated data flow diagrams shown in Fig.4.5 and Fig.4.6. In this chapter however, to test the parallel scalability in the most general setting, α, γ no longer follow Eq.3.8. This means that the data communication for BSGD/CSGD is more complex, as shown in Fig.6.7 and Fig.6.8.

To test the scalability of BSGD/CSGD, Iridis 5 computer cluster is used in this section. Iridis 5 is a high performance computer cluster provided by University of Southampton, with each parallel node equipped with 40 CPUs. The previous 2D system matrix $\mathbf{A} \in \mathbb{R}^{72000\times4096}$ is adopted. The detailed partition and α, γ in this section are shown in Table.6.1. It can be seen that the maximum number of selected blocks is 16. As a result, the applied number of parallel nodes in Iridis 5 is set to 1, 2, 4, 8,16 respectively.

Table 6.1: Partition on $\mathbf{A}$ in scalability test

size of $\mathbf{A}$	M	N	α	γ
72000×4096	5	8	$\frac{2}{5}$	1

The mpi4py package is used here to execute the inner-node communications. The FP/BP projections are performed in matrix-vector multiplications using Python's numpy.dot()

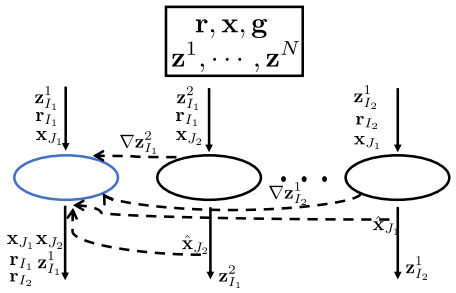
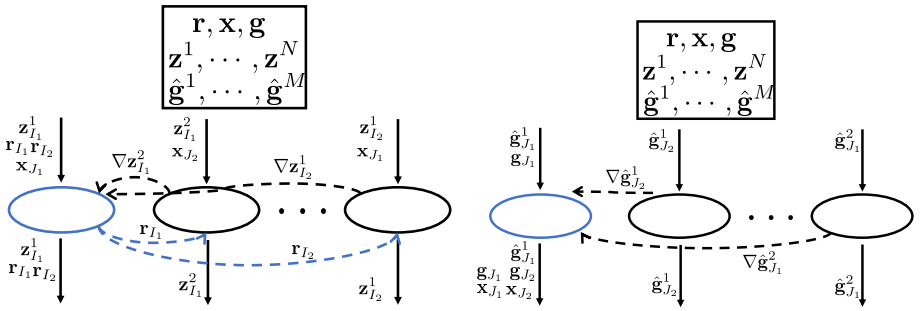


Figure 6.7: The general CSGD work flow. When the computation nodes request data blocks from the master node or send data block back to the master node, a for-loop with point-to-point communication or collective communication can be used. The data blocks are chosen at a totally random manner. In this figure, the root node (blue oval) addresses I_1, J_1 block, the second node addresses I_1, J_2 while the last node address I_2, J_1 data block. Each parallel nodes sends $\nabla \mathbf{z}_I^j$ and the updated $\hat{\mathbf{x}}_J$ to the root node. Since different I_i are used among all nodes, instead of performing a "Reduce" function, the root node performs a "Gather" function to collect all variables and then sequentially updates relative $\mathbf{r}_I$ with different $\nabla \mathbf{z}_I^j$ and the averaged $\mathbf{x}_J$ with different $\hat{\mathbf{x}}_J$.



(a) the first time of communication (b) the second time of communication

Figure 6.8: The data block addressed by each parallel node is the same as Fig.6.7. In BSGD, there are 2 times communication between the root and computation nodes. In the first time of communication, the root node gathers $\nabla \mathbf{z}_I^j$ from all parallel nodes using "Gather" and updates relevant $\mathbf{r}_I$. After this procedure the updated $\mathbf{r}_I$ needs to be sent back to each parallel node. This procedure can be completed by "Scatter" function if a matrix $[\mathbf{r}_{I_1}, \mathbf{r}_{I_1}, \cdots . \mathbf{r}_{I_2}]$ is formed, or is accomplished by a for-loop using point-to-point communication. The update of $\mathbf{g}, \mathbf{x}$ happens in the second time of communication and is similar to the first communication. Attention that because of a more general and random selection method on α, γ, the update of $\mathbf{g}, \mathbf{x}$ only happens on the root node. This is a difference between here and Fig.4.6.

function. To make detailed comparisons of BSGD/CSGD under different scenarios, the percentage of projections over the whole computations is tuned to two different scales. In the first case, the percentage of projections is tuned to be very high. This is achieved by repeatedly generating $\mathbf{A}_I^J$ by slicing $\mathbf{A}$ with index set I, J. Take the BSGD as an example, before the FP $\mathbf{z}_I^j = \mathbf{A}_I^J \mathbf{x}_J$ is performed in each node, $\mathbf{A}_I^J$ is generated by slicing $\mathbf{A}$ (i.e. $\mathbf{A}_I^J = \mathbf{A}[I, J]$ in Python code). When it comes to BP $\hat{\mathbf{g}}_J^i = (\mathbf{A}_I^J)^T \mathbf{r}_I$,

the slicing procedure is repeated again. Although this repeated slicing seems to be redundant because $\mathbf{A}_I^J$ is already stored in parallel nodes after FP and thus can be directly used for the following BP, the repeated generation of $\mathbf{A}_I^J$ reflects the actual process happening in mainstream CT toolboxes such as TIGRE and ASTRA because these toolboxes only output results of FP/BP whilst the generated $\mathbf{A}_I^J$ will be deleted internally. In the second case, the percentage of projections is tuned down. This time $\mathbf{A}_I^J$ is still repeatedly generated but the sliced $\mathbf{A}_I^J$ is pre-calculated and stored in a Python list object $\mathbf{A}_{list}$. When FP/BP requires a $\mathbf{A}_I^J$, they will directly sample it from $\mathbf{A}_{list}$, which makes the generating of $\mathbf{A}_I^J$ much faster than slicing methods and thus reduces the percentage of projections over the total reconstruction time. To test the speed-up, different p are set and the code of BSGD/CSGD is shown in Algo.6.1 and Algo.6.2. For simplicity of coding, the master node and the root computation node are the same node.

The speed-up comparisons of CSGD/BSGD and the percentage of each procedure over

Algorithm 6.1 Apply CSGD algorithm in Iridis 5 network

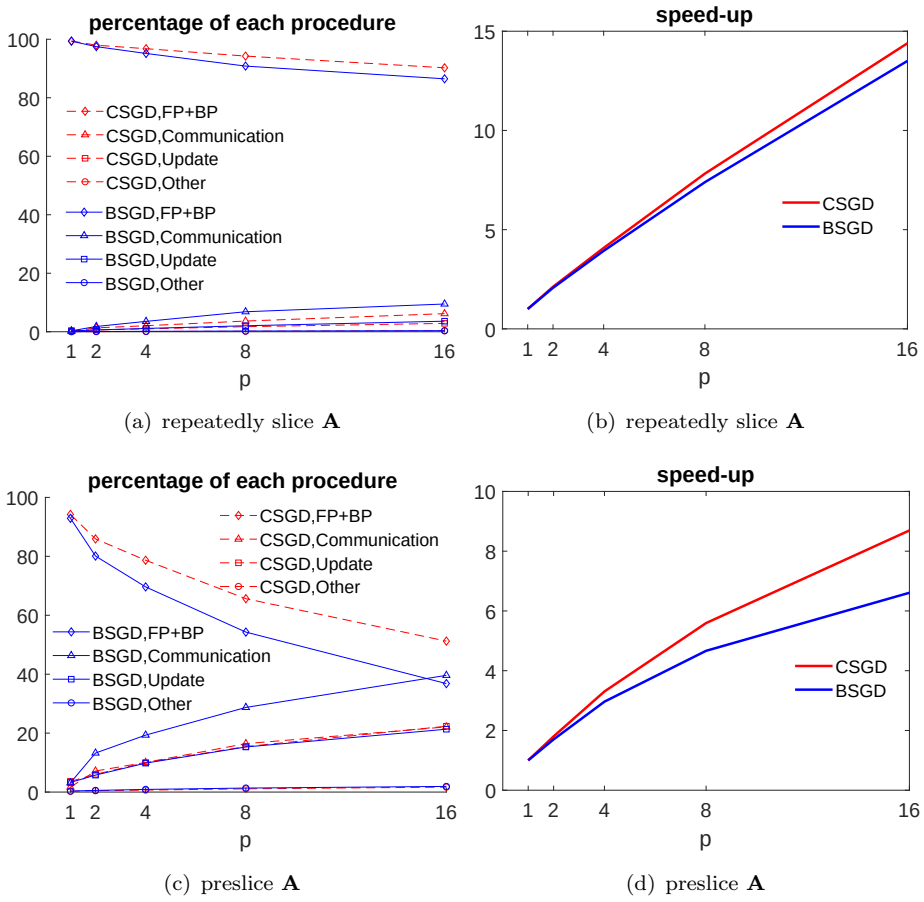
1: Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ is the percentage of the selected row and column blocks respectively. Determine the number of parallel nodes p.
2: **for** $k = 1, 2, \cdots, K_{max}$ **do**
3: Random select αM row blocks from $\{I_i\}_{i\in[1,M]}$ and γN column blocks from $\{J_j\}_{j\in[1,N]}$. Record all i in i_{box} and all j in j_{box}
4: **for** $l = 1, 2, \cdots, \frac{\alpha\gamma MN}{p}$ **do**
5: The root node scatters p $\{\mathbf{z}_{I_i}^j\}_{i\in i_{box}, j\in j_{box}}\{\mathbf{r}_{I_i}\}_{i\in i_{box}}, \{\mathbf{x}_{J_j}\}_{j\in j_{box}}$ to p parallel nodes
6: Delete corresponding i, j from i_{box}, j_{box}
7: All parallel nodes compute $\mathbf{z}_{I_i}^j, \nabla\mathbf{z}_{I_i}^j, \hat{\mathbf{x}}_{J_j}^i$ and send them back to the root node
8: **end for**
9: The root node updates all gathered $\mathbf{z}_{I_i}^j$, updates $\mathbf{x}_{J_j}$ by averaging corresponding gathered $\hat{\mathbf{x}}_{J_j}^i$, updates $\mathbf{r}_{I_i}$ by summing up all gathered $\nabla\mathbf{z}_{I_i}^j$.
10: **end for**
11: $\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

the whole wall time is shown in Fig.6.9. In the figure, "FP+BP" denotes time spent on two ingredients. One is the time spent on matrix-vector multiplication using $\mathbf{A}_I^J$ and $(\mathbf{A}_I^J)^T$, the other is that spent on calculating $\nabla\mathbf{z}_{I_i}^j$ and $\nabla\hat{\mathbf{g}}_{J_j}^i$. "Communication" denotes the time spent on the root node gathering $\nabla\mathbf{z}_{I_i}^j$ and $\nabla\hat{\mathbf{g}}_{J_j}^i$ from other parallel nodes and scattering variables blocks (e.g. $\mathbf{x}_{J_j}$, $\mathbf{r}_{I_i}$, etc.) back to them. "Update" denotes the time spent on the root node to update $\mathbf{r}$,$\mathbf{x}$ and $\mathbf{g}$ in BSGD. While the root node performing update procedure, the other parallel nodes simply keep idle status. The "Other" denotes time spent on two ingredients. The first is time on the random selection process to determine which row/column blocks will be selected in the next epoch. The second is time spent on assembling a new matrix for the latter scatter process. For example, if the root node scatters $\mathbf{x}_{J_1}$ to process 0 and $\mathbf{x}_{J_2}$ to process 1. It

Algorithm 6.2 Apply BSGD algorithm in Iridis 5 network

Initialization: Determine the maximum allowed epoch number K_{max}. Partition row and column indices into sets $\{I_i\}_{i\in[1,M]}$ and $\{J_j\}_{j\in[1,N]}$, $\{\mathbf{x}_{J_j}\}_{j\in[1,N]} = \mathbf{0}$, $\{\mathbf{z}^j\}_{j\in[1,N]} = \mathbf{0}$, $\{\hat{\mathbf{g}}^i\}_{i\in[1,M]} = \mathbf{0}$ and $\mathbf{r} = \mathbf{y}$. α and γ is the percentage of the selected row and column blocks respectively. Determine the number of parallel nodes p.
for $k = 1, 2, \cdots, K_{max}$ **do**
  Random select αM row blocks from $\{I_i\}_{i\in[1,M]}$ and γN column blocks from $\{J_j\}_{j\in[1,N]}$. Record all i in i_{box} and all j in j_{box}
  for $l = 1, 2, \cdots, \frac{\alpha\gamma MN}{p}$ **do**
    The root node scatters p $\{\mathbf{z}^j_{I_i}\}_{i\in i_{box}, j\in j_{box}}, \{\mathbf{x}_{J_j}\}_{j\in j_{box}}$ to p parallel nodes
    Delete corresponding i, j from i_{box}, j_{box}
    All parallel nodes compute $\mathbf{z}^j_{I_i}, \nabla\mathbf{z}^j_{I_i}$ and send them back to root node
  end for
  The root node updates all gathered $\mathbf{z}^j_I$, updates $\mathbf{r}_I$ by summing up all gathered $\nabla\mathbf{z}^j_I$.
  for $l = 1, 2, \cdots, \frac{\alpha\gamma MN}{p}$ **do**
    The root node scatters p $\{\hat{\mathbf{g}}^i_{J_j}\}_{i\in i_{box}, j\in j_{box}}, \{\mathbf{r}_{I_i}\}_{i\in i_{box}}$ to p parallel nodes
    All parallel nodes compute $\hat{\mathbf{g}}^i_{J_j}, \nabla\hat{\mathbf{g}}^i_{J_j}$ and send them back to root node
  end for
  The root node updates all gathered $\hat{\mathbf{g}}^i_{J_j}$, updates $\mathbf{g}_J$ by summing up all gathered $\nabla\hat{\mathbf{g}}^i_J$ and updates $\mathbf{x}_J$ using updated $\mathbf{g}_J$.
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

is necessary to build $\mathbf{x}_{scatter} = [\mathbf{x}_{J_1}, \mathbf{x}_{J_2}]$. This two procedures are both executed on the root node. Similarly, when the root node is performing codes, the other parallel nodes simply keep idle status. Fig.6.9 illustrates two algorithms' scalability where $(a), (b)$ are the case when $\mathbf{A}$ is repeatedly sliced in Python environment and $(c), (d)$ are the case when $\mathbf{A}$ is previously sliced and the sub-matrices $\mathbf{A}^J_I$ are stored in a list object in Python environment. Under both situations, the projection operation in CSGD takes higher a percentage over the whole reconstruction time than in BSGD. This is because CSGD performs 3 matrix-vector multiplications for each $\mathbf{A}^J_I$ whilst the BSGD performs 2. The CSGD shows better scalability than BSGD. This is because the CSGD enjoys a simpler communication scheme than BSGD. It avoids the communication on $\mathbf{g}$ space, whilst the BSGD, as well as the other first order algorithms including SIRT, SAG, etc., has to communicate $\hat{\mathbf{g}}^i_J$ among computation nodes, thus increasing the communication overhead and reducing the percentage of projection part. Repeatedly slicing $\mathbf{A}$ before each projection makes the projection process more time-consuming than pre-slicing $\mathbf{A}$ and thus the projection procedure takes more share among the total reconstruction. This leads to a higher scalability result than pre-slicing $\mathbf{A}$ in both CSGD and BSGD. The figure also shows that in both algorithms, the "FP+BP" and "Communication" takes the majority of the percentage of the whole reconstruction period. Despite that the assembling matrix part in "Other" seems to be time-consuming, in 2D simulations, it actually takes a minimal percentage.



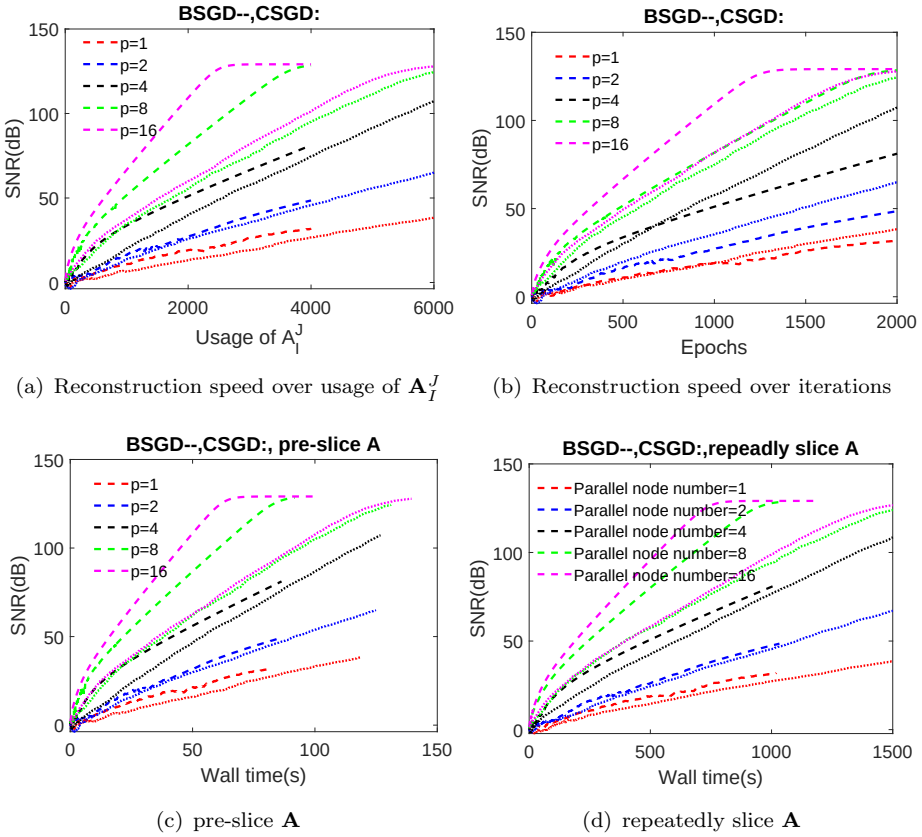
(a) repeatedly slice $\mathbf{A}$ (b) repeatedly slice $\mathbf{A}$

(c) preslice $\mathbf{A}$ (d) preslice $\mathbf{A}$

Figure 6.9: The x-axil is the parallel node number used in simulations. y-axil is the spending time percentage of each procedure takes or the speed up measurement. CSGD has better scalability than BSGD and is more suitable for parallel networks because of the simpler communication scheme.

Although CSGD has better scalability, this does not mean that CSGD can reconstruct the image to a predefined precision by using less wall time than BSGD. To compare the actual speed between the two algorithms, the above partitioning method and the range of p are again used here. This time α, γ are changed according to different p by following Eq.3.8. In each case the parameters were well tuned to ensure the best performance. Reconstruction speeds are shown in Fig.6.10. In the figure, different colours stand for different p situations. Dashed lines represent BSGD and dotted lines represent CSGD. Fig.6.10(a) shows that BSGD is of faster reconstruction speed based on the usage of $\mathbf{A}_I^J$. Fig.6.10(b) shows that CSGD is faster than BSGD in terms of the reconstruction speed over epochs, especially when the number of parallel nodes is smaller than 4. However, each iteration in CSGD contains 3 times of the usage of $\mathbf{A}_I^J$ whilst only 2 matrix-vector multiplications are included in each BSGD's iteration. It means that performing one epoch in CSGD is more time-consuming than BSGD. Fig.6.10(c) and (d) show that regardless of the slicing method on $\mathbf{A}$, the BSGD shows a faster reconstruction speed over the actual wall time, despite the fact that CSGD shows better scalability than

BSGD due to smaller communication overhead.



(a) Reconstruction speed over usage of $\mathbf{A}_I^J$ (b) Reconstruction speed over iterations

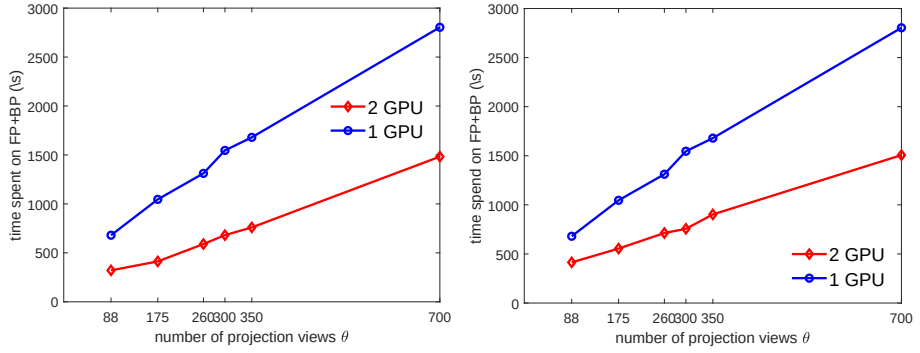
(c) pre-slice $\mathbf{A}$ (d) repeatedly slice $\mathbf{A}$

Figure 6.10: Reconstruction speed comparisons of BSGD and CSGD in realistic parallel network.

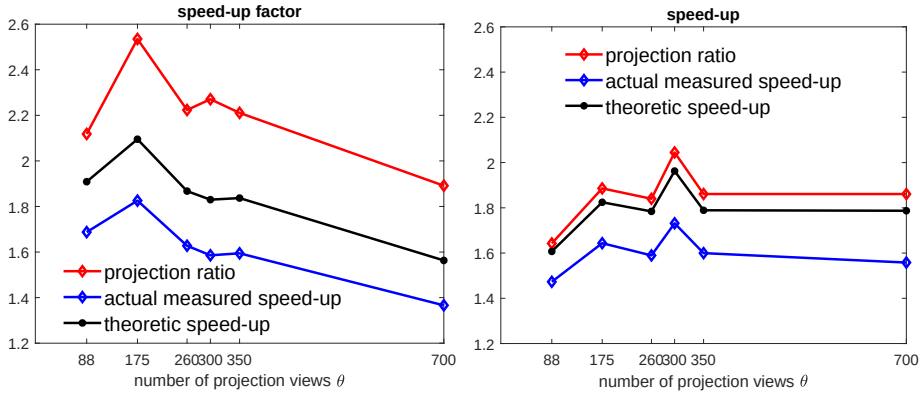
6.3 Synchronous application of BSGD in GPU nodes

BSGD has shown better performance in terms of reconstruction speed and as a result, it is applied in a realistic Iridis node which is equipped with four GTX 1080Ti GPUs to reconstruct a 3D cone beam scanning data. In this section, the data communication in BSGD is the same as Algo.6.2. The only difference is that the FP and BP process no longer use CPU to calculate matrix-vector multiplications but use GPU-accelerated function provided in Python form TIGRE toolbox. Previous 2D simulations show that assembling a new matrix before scattering data blocks takes minimal percentage among the whole time. However, this phenomena does not hold in large-scale 3D cases. To verify this, BSGD is applied in one and two GPUs within one Iridis node respectively to compare the speed-up factor under two different data communication schemes. One scheme is to use collective communications including "scatter" and "gather" to send and collect different data blocks to different process. This requires to assemble a new matrix before the communication. The other method is to use point-to-point communications with a

"for-loop" to connect each separate node with the root node. To ensure the computation resources are fair, each process is assigned with 10 CPUs and 1 GPU. The image is set as a 512^3 head phantom and the detector is a 1024^2 square panel. The size of voxel in 3D volume and pixel in the detector are both $1mm$. OP and OD in Fig.3.5 is $1500mm$ and $1000mm$ respectively. The detector scans the image for a contact circle and the number of projection views θ is set from 88 to 700. (When the number of θ becomes even larger, the python function Gather and Scatter cannot work due the transferred data size exceed the buffer size.) The partition is determined: $M = 1, N = 4, \alpha = \gamma = 1$, and the BSGD is executed for 100 epochs for both communication schemes. The speed-up comparisons under two communication schemes are shown in Fig.6.11. In the Fig.6.11, the projec-



(a) Communication type: collective (b) Communication type: point-to-point



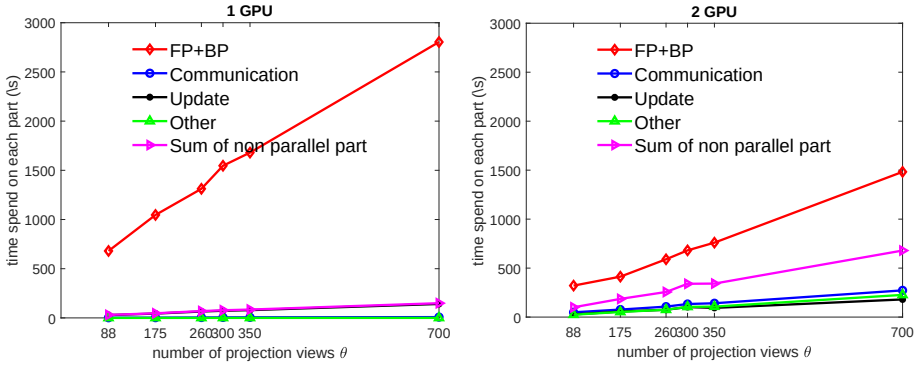
(c) Communication type: collective (d) Communication type: point-to-point

Figure 6.11: (a) and (b) are the time spent on GPU calculations under different numbers of θ. (c) and (d) are theoretical and realistic speed-up comparisons.

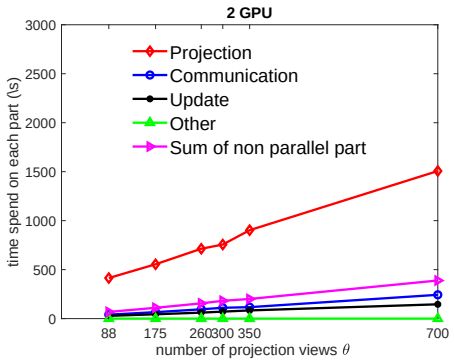
tion ratio is the ration of spent time on "FP+BP" when using different number of GPUs (i.e. blue lines divided by red lines in Fig.6.11(a) and Fig.6.11(b)). The actual measured speed-up is the actual ratio of wall time when using different numbers of GPUs. The theoretic speed-up is calculated as if the communication is finished instantly (i.e. it is calculated as $\frac{\text{total time when using 1 GPU}}{\text{total time when using 2 GPUs - total communication time}}$). Fig.6.11(a)-(b) have demonstrated that when using two GPUs simultaneously, the time spent on projection part is reduced compared with using one GPU case. However, the time is not exactly

halved, as shown in "projection ratio" in Fig.6.11(c)-(d). This is because the computation speed of TIGRE in Python environment currently is unstable. Fig.6.11(c)-(d) have shown that there is a gap between projection ratio and theoretic speed-up. This is because the existing "Update" and "Other" procedure mentioned above. It can be seen that this gap in collective situation is larger that than that in point-to-point situation, which suggests that the share of "Other" in collective communication is larger than point-to-point communication.

The time spent on different parts is shown in Fig.6.12. The "Sum of non parallel part" is



(a) (b) Communication type: collective



(c) Communication type: point-to-point

Figure 6.12: Time spent on each procedure in BSGD when using two different communication schemes Point-to-point communication scheme is more suitable than collective communication scheme in large 3D dataset because of a smaller "Other" part.

the sum of time on "Communication", "Update" and "Other" during 100 epochs. When using one GPU, the code is completely sequential and projections takes the majority part of total spent wall time. When using two GPUs, since two GPUs perform the projection simultaneously, the total spent time on this part is significantly reduced compared with one GPU case. However, the communication cost increases and thus increases the 'Sum of non parallel part". The "Other" part in collective communication type is significantly higher than that in point-to-point type. According to the definition of "Other" in section 6.2, it means that the assembling a new matrix for collective

communication in 3D cone beam case is time-consuming and thus the point-to-point communication is more suitable in larger-scale 3D case. As a result, in the following simulations the point-to-point communication scheme is a default setting.

When the BSGD is scaled to 4 GPU situation, each process is also assigned to 10 CPUs and 1 GPU. In this simulations, the physical cubic head phantom is fixed as $256 \times 256 \times 256mm^3$ and the detector physical size is $512 \times 512mm^2$. OP and OD in Fig.3.5 is $1500mm$ and $1000mm$ respectively. A parameter S is defined, and the scanned phantom contains S^3 voxels and the square detector contains S^2 pixels. The point source and detector rotate around the volume horizontally for a full circle through S projection views. To simplify the comparison under different problem sizes, $M = 1, N = 4, \alpha = \gamma = 1$. Under different sizes S, the speed-up, time spent on each part and their separate percentage in the reconstruction time is shown in Fig.6.13, Fig.6.14 and Fig.6.15. In all scenarios BSGD is run for 100 epochs. Fig.6.13 experimentally shows that for a single node with several GPUs, the speed up is about 1.73 when 2 GPU are used and 2.94 when 4 GPU are used. When the size increases, especially from S =128 to 800, the speed-up factor remains stable. The reason why the speed-up is various is the computation speed of TIGRE is unstable in Python environment. Besides, the data communication speed is also experimentally found to be unstable. Fig.6.14 experimentally shows the time spent on each procedure in BSGD under different numbers of GPUs. Fig.6.14(c) shows that the communication cost increases faster along with S than BP. This is because the TIGRE toolbox uses a simplified computation method to approximate the matrix-vector multiplication in BP, thus the computation speed of BP is accelerated. A detailed discussion about the approximations used in TIGRE to efficiently compute BP is beyond the scope of this thesis. From the multi-CPU and hybrid CPU-GPU simulation results, it is however worth mentioning that the approximations used in TIGRE do not seem to affect the reconstruction quality achieved with BSGD. We are here interested in the use of the method for large scale problems where the cost of matrix vector products dominates and thus dominates how the algorithm scales. In this setting, the other first-order algorithms will also have the same communication cost as BSGD and they thus scale similarly to BSGD. Fig.6.15 further demonstrates that when 4 GPUs are used, the communication overhead takes larger share than 2 GPU case, taking nearly 40% among the total wall time. The fast increasing communication overhead suggests that the parallel scalability of BSGD when using TIGRE toolbox is limited. However, it is worth mentioning that this drawback is common for all other first-order algorithms since they all have similar communication schemes.

6.4 A flexible parallel application of BSGD

In this section, BSGD is further researched and is applied in a more flexible parallel way to further increase its reconstruction speed when computation nodes are of different

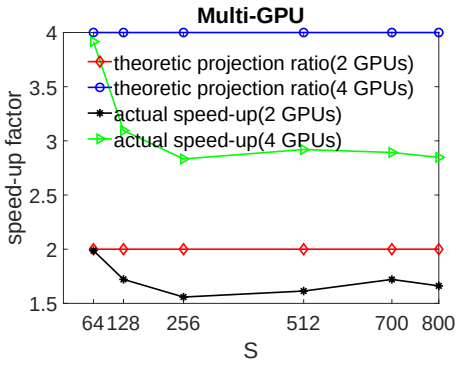
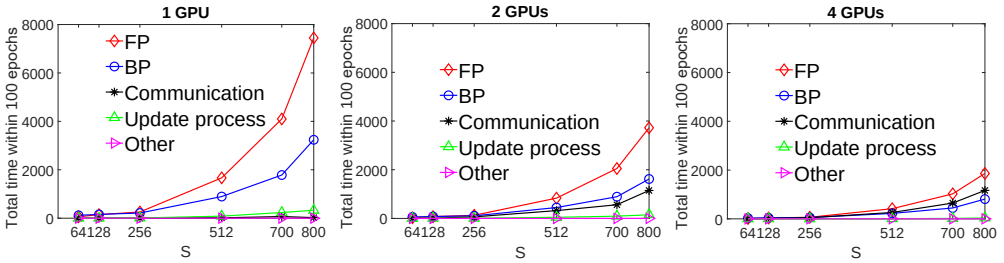


Figure 6.13: Scalability of BSGD when used in multi-GPU network in 3D scanning geometry. When the size increases, especially from S =128 to 800, the speed-up factor remains stable.



(a) (b) (c)

Figure 6.14: When the size gradually increases, especially from S =128 to 800, the time spent on FP increases faster than the other cost. However in the 4 GPU setting, communication costs are larger than those for BP. This is due to the efficient implementation of BP in TIGRE, which allows this to be computed much faster than its counterpart FP. This means that the scalability of BSGD is limited in TIGRE environment.

computation speeds. It does not require all parallel nodes to wait for each other until all of them have finished computations. Instead, as long as there is a node that has finished computation, then the root node immediately communicates with it to update the corresponding variables. This is an asynchronous communication and so we call this version of the algorithm Asynchronous BSGD(ABSGD). Two versions of ABSGD have been proposed and studied in detail.

The ABSGD is mainly proposed to solve the case when the parallel network suffers from an unforeseen network delay or when one computation node is significantly slower than the others. However, these two situations are difficult to simulate when the Iridis 5 network is used since each node in it is of the same computation speed and the communications between nodes are stable for the most of time. It is possible to artificially introduce some delays for some nodes or during communications, but this makes the whole simulations rather time-consuming and inefficient. As a result, the simulations in this section mainly is performed in MATLAB serial code and a virtual wall time t is introduced to replace the previous time reference "Usage of $\mathbf{A}_I^J$". This virtual time t can

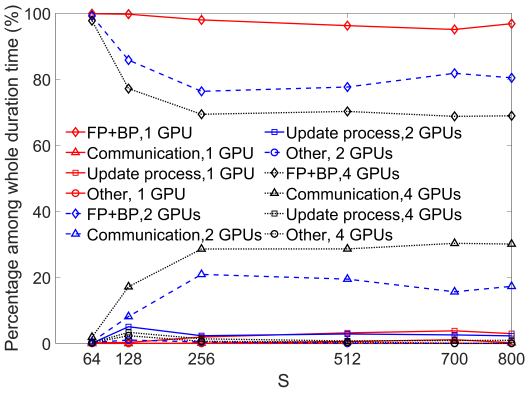


Figure 6.15: Percentage of spent time on each procedure in BSGD under different numbers of GPUs. Once a specific problem size has been reached, then the relative percentage does not change significantly for increasing problem sizes.

be used to reflect different delays in each separate node and communications without loosing the efficiency of code execution. Besides, it also makes the simulation results stable and repeatable.

6.4.1 Redefining BSGD time counting system

The previous parallel application of BSGD used synchronous communication. The root node does not perform the "Gather" function to collect information from all parallel nodes until all computation nodes have finished their computations. In all simulations shown in Chapters 3,4 and 5, the computation nodes were assumed to have the same computation ability and they receive the data at the same time. This assumption means that all parallel nodes start and finish the computation at the same time, thus the "usage of $\mathbf{A}_I^J$", which is proportional to the time spent on data communication and the data projection, can be used as a time reference. However, sometimes, the computation nodes may not have the same computational abilities or the parallel node cannot receive the data or start the projection at the same time. This phenomenon has been implied in the demonstrations of BSGD/CSGD general communication frames shown in Fig.6.7 and Fig.6.8. At the data distribution and collection stages, if the master node does not use collective communication functions such as "Scatter,Bcast,Gather,Reduce" but use a for-loop with point-to-point communication functions "Send,Recv", then different nodes will receive data block at different times and the first node receiving the data block will also be the first node to finish its calculation and thus becomes the fastest node. Similarly, the last node receiving a data block is the slowest node. In this case, the fast computation nodes have to wait for the slow computation nodes to finish their calculations. This communication property makes the fact that the reconstruction speed is influenced by the slowest computation node. The existence of waiting time makes "the usage of $\mathbf{A}_I^J$" no longer reflect the actual wall time. As a result, here a virtual wall time t is introduced to be used as a time reference reflecting the reconstruction speed. For

each parallel node, t is mainly composed of: (1) communication overhead $\{t_{commu,l}\}$, (2) projection time $\{t_{proj,l}\}$ (3) waiting time $\{t_{wait,l}\}$, where $l \in \{0, 1, \cdots, p-1\}$ is the node index among all p parallel nodes. In addition, for the maser node or the root node, there is an added time: update time $\{t_{upd}\}$, which is the time spent on updating $\mathbf{r}, \mathbf{g}, \mathbf{x}$.

A time counting algorithm for BSGD is shown in Algo.6.3. In the algorithm, the master node sends/gathers data to/from parallel node using a cyclic point-to-point communication scheme. The master node performs the update of $\mathbf{r}, \mathbf{g}, \mathbf{x}$. The parallel node only performs projections and different computation nodes have different computation abilities. For simplicity, here we only discuss the case when the selected block number exactly equals the number of parallel nodes p.

Algorithm 6.3 BSGD, a virtual wall time t is introduced

Initialization: All initialization condition is unchanged. t is a virtual wall time on the master node. $\{\tilde{t}_l\}_{l\in[0,p-1]}$ is the "predicted" wall time for each computation node when they finish current projection. t_{other} is the time spent on selecting different blocks, which is a tiny amount compared with projection and communication.
$t = 0$
for $k = 1, 2, \cdots, K_{max}$ **do**
  Random select αM row blocks from $\{I_i\}_{i\in[1,M]}$ and γN column blocks from $\{J_j\}_{j\in[1,N]}$. Record all selected i in i_{box} and all selected j in j_{box}
  $t = t + t_{other}$ (t_{other} can be ignored since it is very small)
  for $l = 0, 1, \cdots, p-1$ **do**
    The root node sends $\{\mathbf{z}^j_{I_i}\}_{i=i_{box}[l],j=j_{box}[l]}, \{\mathbf{x}_{J_j}\}_{j=j_{box}[l]}$ to l^{th} parallel nodes
    $\tilde{t}_l = t + t_{commu,l}$
    $t = \tilde{t}_l$
  end for
  All parallel nodes compute $\mathbf{z}^j_I, \nabla\mathbf{z}^j_I$
  For l^{th} parallel node $l = [0, 1, \cdots, p-1]$: $\tilde{t}_l = \tilde{t}_l + t_{proj,l}$
  When all parallel nodes finish calculating, root node begin to receive their data in a sequential way and update $\mathbf{r}_I$
  $t = max(\{\tilde{t}_l\}_{l\in[0,p-1]}) + p(t_{commu,l} + t_{upd,\mathbf{r}_I})$
  for $l = 0, 1, \cdots, p-1$ **do**
    The root node sends $\{\hat{\mathbf{g}}^i_{J_j}\}_{i=i_{box}[l],j=j_{box}[l]}, \{\mathbf{r}_{I_i}\}_{i=i_{box}[l]}$ to l^{th} parallel nodes
    $\tilde{t}_l = t + t_{commu,l}$
    $t = \tilde{t}_l$
  end for
  All parallel nodes compute $\hat{\mathbf{g}}^i_{J_j}, \nabla\hat{\mathbf{g}}^i_{J_j}$ and send them back to root node
  For l^{th} parallel node $l = [0, 1, \cdots, p-1]$: $\tilde{t}_l = \tilde{t}_l + t_{proj,l}$
  When all parallel nodes finish calculating, root node begin to receive their data in a sequential way and update $\mathbf{g}_J, \mathbf{x}_J$
  $t = max(\{\tilde{t}_l\}_{l\in[0,p-1]}) + p(t_{commu,l} + t_{upd,\mathbf{g}_J,\mathbf{x}_J})$
end for
$\mathbf{x}_{solution} = [\mathbf{x}_{J_1}, ..., \mathbf{x}_{J_N}]^T$

In simulations, $t_{commu,l}$, $t_{proj,l}$ and t_{upd}, are calculated by multiplying the size of data blocks with parameters τ_{commu}, $\tau_{proj,l}$ and τ_{upd} respectively. For example, an $\mathbf{x}_J$ with

size n is sent to the l^{th} parallel node and $\mathbf{A}_I^J \mathbf{x}_J$ ($\mathbf{A}_I^J \in \mathbb{R}^{m\times n}$) is performed in this node, then $t_{commu,l} = \tau_{commu} n, t_{proj,l} = \tau_{proj} mn$. To simplify the parameter tuning, in simulations, τ_{proj} is set as different values for the l^{th} node to reflect that different computation nodes have different computation abilities and $\tau_{commu,l}, \tau_{upd}$ is set as a constant values. Lines 14 and 23 in Algo.6.3 suggest that for each iteration, the fastest nodes have to wait for the slowest node, meaning that the synchronous BSGD does not fully exploit the computational advantage of fast nodes.

6.4.2 Two asynchronous BSGD forms

Here two forms of "asynchronous" BSGD (ABSGD) are proposed, called ABSGD-Ver1 and ABSDG-Ver2 respectively. They both do not require the fast nodes to wait for the slowest node and can communicate with the master node when they are ready. ABSGD-Ver1 is shown in Algo.6.4.

In Algo.6.4. The master node first sends data blocks to all parallel nodes as BSGD does. After the sending process, the master node checks whether there is any computation node that has already finished computation and is waiting for the master to communicate the data. If there is no waiting computation node then the master node simply waits until there is one node that has finished its computation (shown in Algo.6.4 line 26-27). Otherwise, if there are computation nodes that have finished the computations after the master sending all data, (shown in Algo.6.4 line 29-30), the algorithm then finds all waiting nodes and the maser node is going to communicate with them. By only addressing the waiting nodes or the fastest node, the master node can perform more updates of $\mathbf{r}, \mathbf{g}$ and $\mathbf{x}$ than synchronous BSGD within the same virtual wall time t, thus can have a faster reconstruction speed. In addition, the waiting time for each node can be significantly reduced compared to synchronous BSGD.

In Algo.6.4 line 32, if $length(\tilde{i}) > 1$, then some nodes still have to wait since only one computation node can communicate with the master node. This part's waiting time is further reduced in ABSDG-Ver2 by only communicating with one waiting node each time (i.e. enforcing $\tilde{i}$ to only contain 1 element). A while loop to repeatedly check whether there is a waiting node is proposed, as shown in Algo.6.5. This function is used by master node. It can be seen that there is only one node is allowed to communicate with the master node and when the master node finish updating $\mathbf{r}$ or $\mathbf{g}, \mathbf{x}$, the master node immediately sends data back to this computation node, thus avoiding the computation node waiting for the other node which is sending data to the master node. With *Check* function, the ABSGD-Ver2 is shown in Algo.6.6.

Algorithm 6.4 ABSGD-Ver1

Initialization: Same as Algo.6.3.
for $k = 1, 2, ..., K_{max}$ **do**
  if $k == 1$ **then**
    Random select αM row blocks and γN column blocks. Record all selected i in i_{box} and all selected j in j_{box}
    $t = t + t_{other}$
    for $l = 0, 1, \cdots, p-1$ **do**
      The root node sends $\{\mathbf{z}^j_{I_i}\}_{i=i_{box}[l], j=j_{box}[l]}, \{\mathbf{x}_{J_j}\}_{j=j_{box}[l]}$ to l^{th} parallel nodes
      $\tilde{t}_l = t + t_{commu,l}$
      $t = \tilde{t}_l$
    end for
    All parallel nodes compute $\mathbf{z}^j_I, \nabla \mathbf{z}^j_I$
    Each node: $\tilde{t}_l = \tilde{t}_l + t_{proj,l}$
  else
    for $l = 1, 2, length(\tilde{i})$ **do**
      if last time the node $\tilde{i}(l)$ has performed FP **then**
        The root node sends $\hat{\mathbf{g}}^i_{J_j}, \mathbf{r}^j_{I_i}$ to node $\tilde{i}(l)$, where i, j is the same pair with the case when node $\tilde{i}(l)$ performed the FP. This time node $\tilde{i}(l)$ performs BP.
      else if last time the node $\tilde{i}(l)$ has performed BP **then**
        The root node sends new $\mathbf{z}^j_{I_i}, \mathbf{x}_{J_j}$ to node $\tilde{i}(l)$, where i, j is a new pair that is different from all pairs in other parallel nodes. This time node $\tilde{i}(l)$ performs FP.
      end if
      $\tilde{t}_{\tilde{i}(l)} = t + t_{commu,\tilde{i}(l)}$
      $t = \tilde{t}_{\tilde{i}(l)}$
      $\tilde{t}_{\tilde{i}(l)} = \tilde{t}_{\tilde{i}(l)} + t_{proj,\tilde{i}(l)}$
    end for
  end if
  if $t < min(\{\tilde{t}_l\}_{l\in[0,p-1]})$ **then**
    Find the node with the smallest $\tilde{t}$, name this node as node $\tilde{i}$ and transfer data from node $\tilde{i}$ to root node
    $t = \tilde{t}_{\tilde{i}} + t_{commu}$
  else
    Find elements within $\{\tilde{t}_l\}_{l\in[0,p-1]}$ that is smaller than t, assemble all these nodes' index number into one set $\tilde{i}$ and transfer data from those node $\tilde{i}$ to root node
    $t = t + \sum_{l\in\tilde{i}} t_{commu,\tilde{i}(l)}$
  end if
  for $l = 1, 2, \cdots, length(\tilde{i})$ **do**
    Root node updates $\mathbf{r}_I$ or $\mathbf{g}_J, \mathbf{x}_J$ according to data in node $\tilde{i}(l)$
    $t = t + t_{upd}$
  end for
end for

Algorithm 6.5 Check$(t, \{\tilde{t}_l\}_{l\in[0,p-1]})$

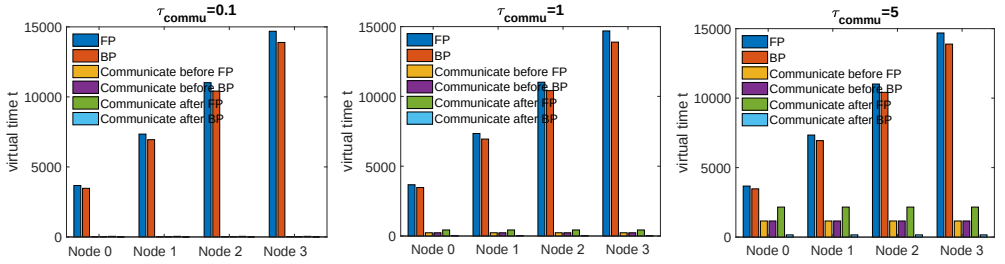
Initialization: $flag = 1$
while $flag$ **do**
  $\tilde{t}_{min} = min(\{\tilde{t}_l\}_{l\in[0,p-1]})$.
  if $t >= \tilde{t}_{min}$ **then**
    Find the node with the smallest $\tilde{t}$, name this node as node $\tilde{i}$
    Transfer data from node $\tilde{i}$ to root node
    $t = t + t_{commu,\tilde{i}}$
    The root node performs update.
    $t = t + t_{upd}$
    The root node sends proper data block back to node $\tilde{i}$. Reference Algo.6.4 in terms of selecting "proper" data block
    $t = t + t_{commu,\tilde{i}}$
    Node $\tilde{i}$ performs projection
    $\tilde{t}_{\tilde{i}} = t + t_{proj,l}$
  else
    $flag = 0$
  end if
end while

Algorithm 6.6 ABSGD-Ver2, a virtual wall time t is introduced

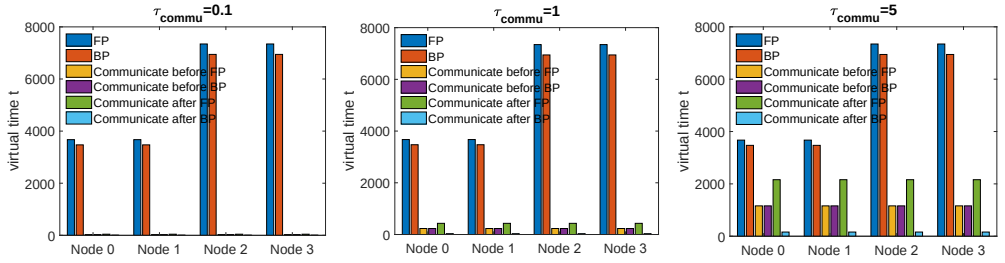
Initialization: Same as Algo.6.3.
for $k = 1, 2, \cdots, K_{max}$ **do**
  if $k == 1$ **then**
    Random select αM row blocks and γN column blocks. Record all selected i in i_{box} and all selected j in j_{box}
    for $l = 0, 1, \cdots, p-1$ **do**
      The root node sends $\{\mathbf{z}_{I_i}^j\}_{i=i_{box}[l],j=j_{box}[l]}, \{\mathbf{x}_{J_j}\}_{j=j_{box}[l]}$ to l^{th} parallel nodes
      $\tilde{t}_l = t + t_{commu,l}$
      $t = \tilde{t}_l$
      l^{th} parallel node does projection
      $\tilde{t}_l = \tilde{t}_l + t_{proj,l}$
    end for
    Check$(t, \{\tilde{t}_l\}_{l\in[0,p-1]})$
  end if
  Find the node with the smallest $\tilde{t}$, name this node as node $\tilde{i}$
  Transfer data from node $\tilde{i}$ to root node and the root node performs update
  $t = \tilde{t}_{\tilde{i}} + t_{commu} + t_{upd}$
  The root node sends data back to node $\tilde{i}$
  $t = t + t_{commu}$
  Node $\tilde{i}$ performs projections
  $\tilde{t}_{\tilde{i}} = t + t_{proj,\tilde{i}}$
  Check$(t, \{\tilde{t}_l\}_{l\in[0,p-1]})$
end for

6.4.3 Simulations to compare BSGD and ABSGD

To verify the properties of ABSGD algorithms, a tiny CT scanning system where $\mathbf{A} \in \mathbb{R}^{1080\times256}$ is divided into $M = 5, N = 16$ blocks and a parallel node with 4 computation nodes ($p = 4$) are adopted. The projection data are stored in double-precision floating-point format and there is no added noise on it. All algorithms are stopped when the virtual time t achieves 4×10^8. During this process the trend of SNR over the virtual time t can reflect convergence properties of different algorithms. τ_{commu} and τ_{proj} are set to different values and τ_{upd} is set as constant 1 for simplicity. The comparisons of the time spent on each projection and communication under different $\tau_{commu}, \tau_{proj}$ is shown in Fig.6.16. In simulations, three different types of τ_{proj} are set. In the first case (the first row of figure), $\tau_{proj,l}$ gradually increases when the node index number increases. It means that the node 0 is the fastest node and the node 3 is the slowest node. As a result, the time spent on FP/BP gradually increases when the node number index l increases. In the second scenario (the second row of figure), the first two nodes ($l = 0, 1$) are relatively fast with $\tau_{proj,l} = 1$ while the last two nodes ($l = 2, 3$) are relatively slow with $\tau_{proj,l} = 2$. For both cases, the range of τ_{commu} covers from 0.1 to 5. The figure have reflected that communication overhead gradually takes larger portion during the reconstruction process. In the third case, $\tau_{proj,l} = 1$, which means that all nodes are of the same computation ability. It is not shown in the figure because all nodes have the same shape with "Node 0" in Fig.6.16(a)-(c).



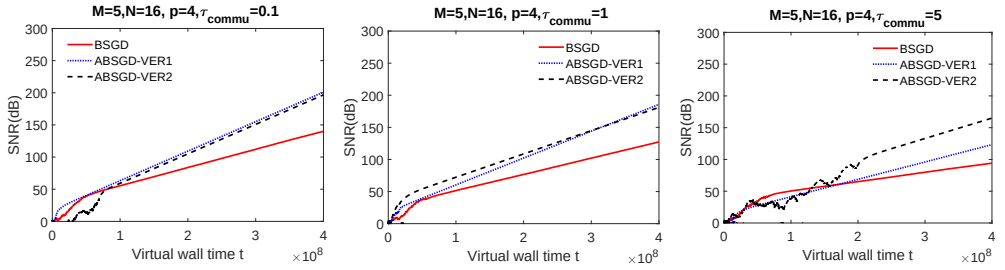
(a) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$ (b) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$ (c) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$



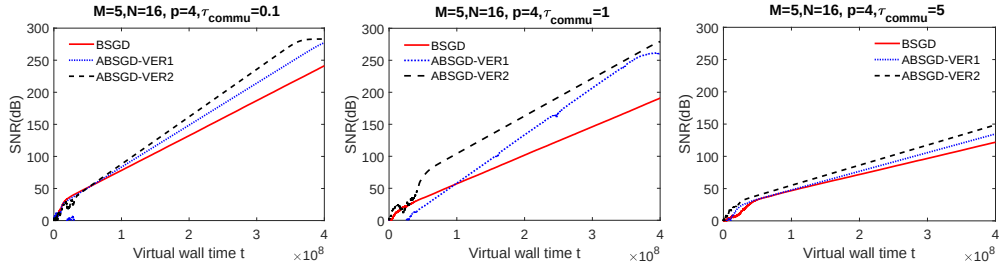
(d) $\tau_{proj,l} = 1(l < 2)or2(l > 1), l = 0, \cdots, p-1$ (e) $\tau_{proj,l} = 1(l < 2)or2(l > 1), l = 0, \cdots, p-1$ (f) $\tau_{proj,l} = 1(l < 2)or2(l > 1), l = 0, \cdots, p-1$

Figure 6.16: Illustration of virtual time spent on each node under different communication delays.

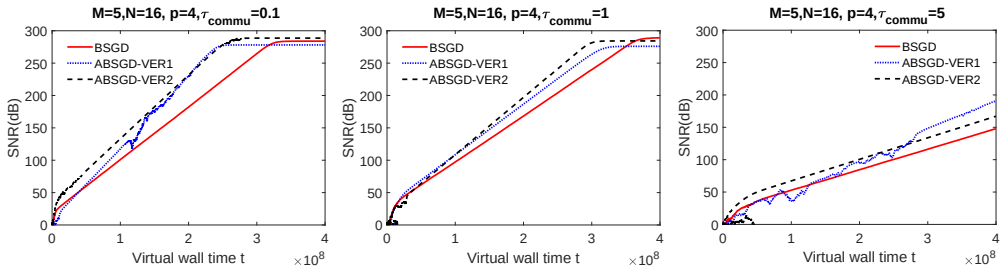
The simulation results of reconstruction speed of BSGD, ABSGD-VER1, and ABSGD-Ver2 are shown in Fig.6.17. In the figure, the red-solid lines are BSGD results and blue-dotted lines are ABSGD-Ver1 results. The black-dashed lines are the ABSGD-Ver2. All simulations are repeated with different step lengths μ from a tiny value to the largest value ensuring convergence. It can be seen that under first two cases of τ_{proj} setting (first two rows), ABSGD-Ver1 and ABSGD-Ver2 both outperform the synchronous BSGD form. In the third case of τ_{proj} setting (third row) where each node is of the same computation capacity, when the communication overhead is tiny ($\tau_{commu} = 0.1$), each node can be viewed that they receive the data block at the same time and ABSGD barely outperforms BSGD. However, when τ_{commu} increases to 1 and 5, different node receives the data block at different time, and then ABSGD gradually shows the advantage over BSGD. The fact that both ABSGD reaches to over 100 dB suggests that the asynchronous application of BSGD is able to reach to high accuracy solutions in realistic applications. The ABSGD-Ver2 is more recommended since it has a simpler communication scheme than ABSGD-Ver1. Besides, it can even outperform the ABSGD-Ver1 especially when τ_{commu} is 5.



(a) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$ (b) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$ (c) $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$



(d) $\tau_{proj,l} = 1(l < 2) or 2(l > 1), l = 0, \cdots, p-1$ (e) $\tau_{proj,l} = 1(l < 2) or 2(l > 1), l = 0, \cdots, p-1$ (f) $\tau_{proj,l} = 1(l < 2) or 2(l > 1), l = 0, \cdots, p-1$



(g) $\tau_{proj,l} = 1(l = 0, \cdots, p-1)$ (h) $\tau_{proj,l} = 1(l = 0, \cdots, p-1)$ (i) $\tau_{proj,l} = 1(l = 0, \cdots, p-1)$

Figure 6.17: Reconstruction speeds of synchronous BSGD and ABSGD under three different computation delays.

The main reason why the ABSGD can outperform BSGD is that the waiting time of computation node is reduced, thus enabling the fast node to perform more projections to exploit the fast computation ability. Each node's waiting time is calculated as the total wall time t minus each node's projection time and communication time. The waiting time of each node through different simulations are shown in Fig.6.18, Fig.6.19 and Fig.6.20.

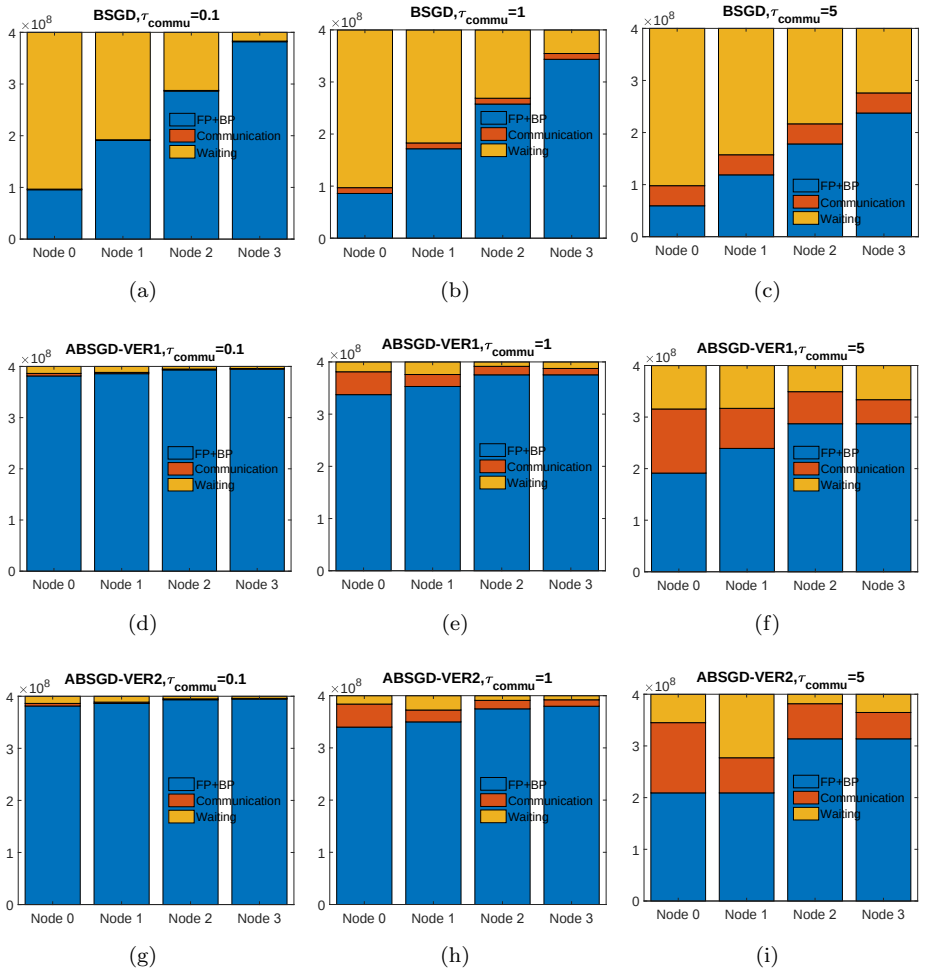


Figure 6.18: The relative time distribution on each part. This figure reflects the case when $\tau_{proj,l} = l + 1, l = 0, \cdots, p-1$. Different row represents different method's relative distribution of spending time under different communication overheads. It can be seen that the increase of τ_{commu} increases the waiting time and the communication overhead for each node. Each column provides comparisons of different methods under the same τ_{commu}. It can be seen that ABSGD-Ver1 and ABSGD-Ver2 both significantly reduce the waiting time and thus increase the projection time compared with BSGD. It means that the asynchronous BSGD can perform more projections than BSGD within the same wall time, leading to more updates on $\mathbf{x}$. This explains why both ABSGD is faster than BSGD within the same wall time. When $\tau_{commu} = 5$, the ABSGD-Ver2 can further reduce the waiting time compared with ABSGD-Ver1. This is due to the algorithm's property which only processes one computation node each time.

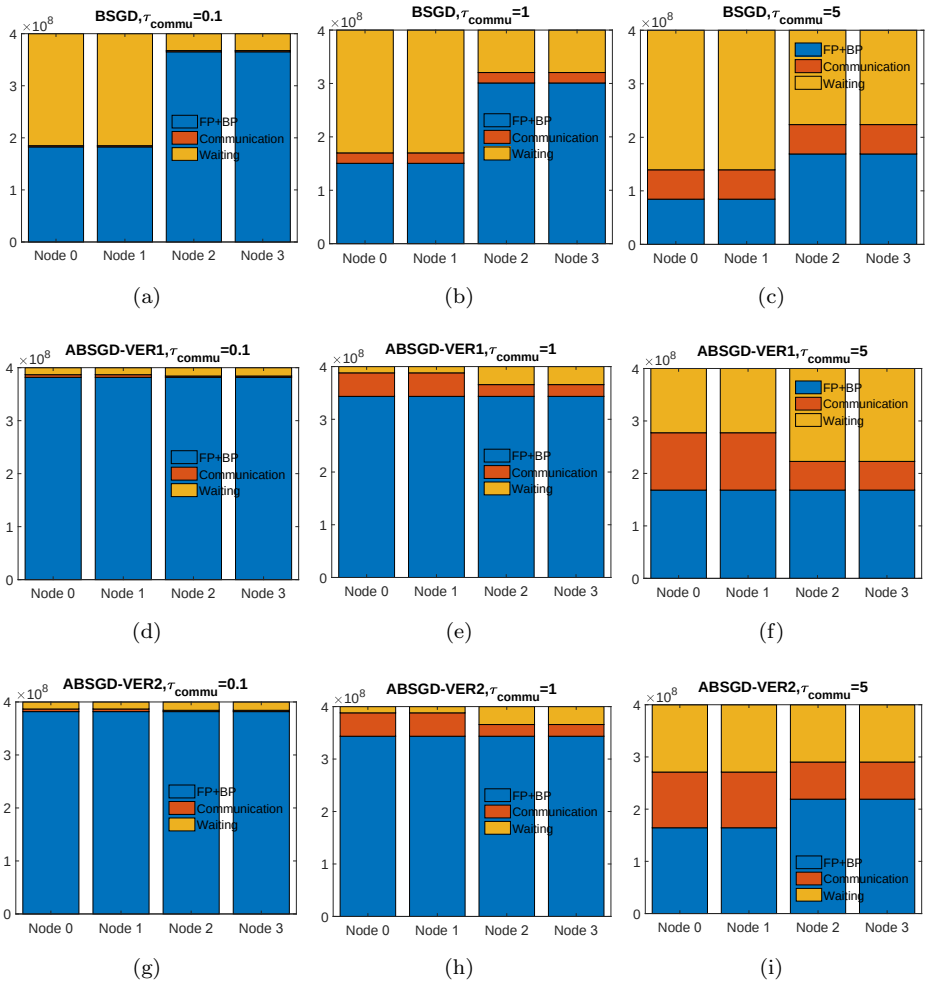


Figure 6.19: This figure reflects relative time distribution when $\tau_{proj,l} = 1(l < 2)$ or $2(l > 1)$ ABSGD shows the same trend with Fig.6.18. Two ABSGD algorithms can increase the projection time while reducing the waiting time, thus increasing the reconstruction speed.

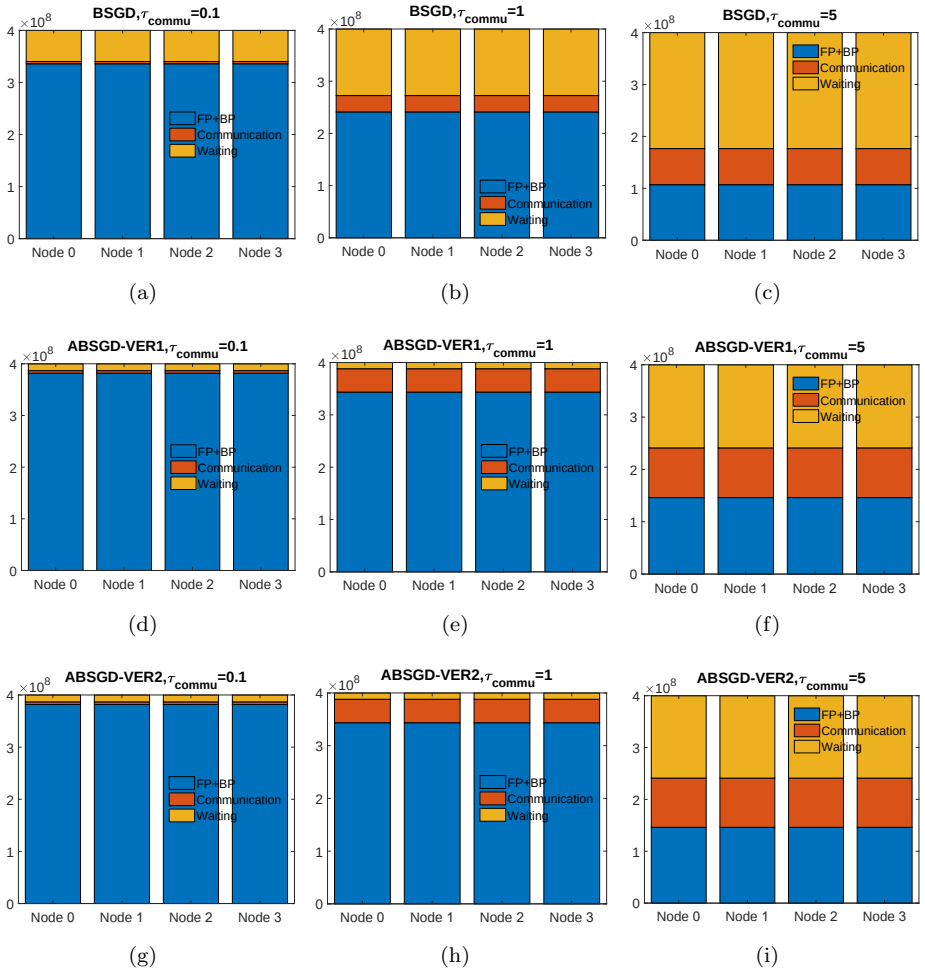


Figure 6.20: This figure reflects relative time distribution when $\tau_{proj,l} = 1, l = 0, \cdots, p-1$. Two ABSGD algorithms can increase the projection time while reducing the waiting time, thus increasing the reconstruction speed. It is worth mentioning that when $\tau_{commu} = 0.1$, the waiting time in BSGD is not long enough to significantly slow down the reconstruction speed, thus making the ABSGD cannot outperform BSGD even if ABSGD further reduces the waiting time.

Furthermore, it is worth mentioning that the automatic parameter tuning method proposed in the section 4.6 is still applicable to ABSGD. Here I combine the automatic tuning method with ABSGD-VER2. Simulations show that in the toy problem $\mathbf{A} \in \mathbf{R}^{1024\times256}$, when $\tau_{proj,l} = l+1, l = 0, \cdots, p-1$ and $\tau_{commu} = \tau_{upd} = 1$, the automatic parameter tuning strategy with previous choice of $\epsilon, \delta, t_1, t_2$ works well. As shown in Fig.6.21.

6.4.4 ABSGD in non-block communications

From the above MATLAB serial simulations, it can be seen that when the parallel computation nodes have different computational abilities, then ABSGD can outperform

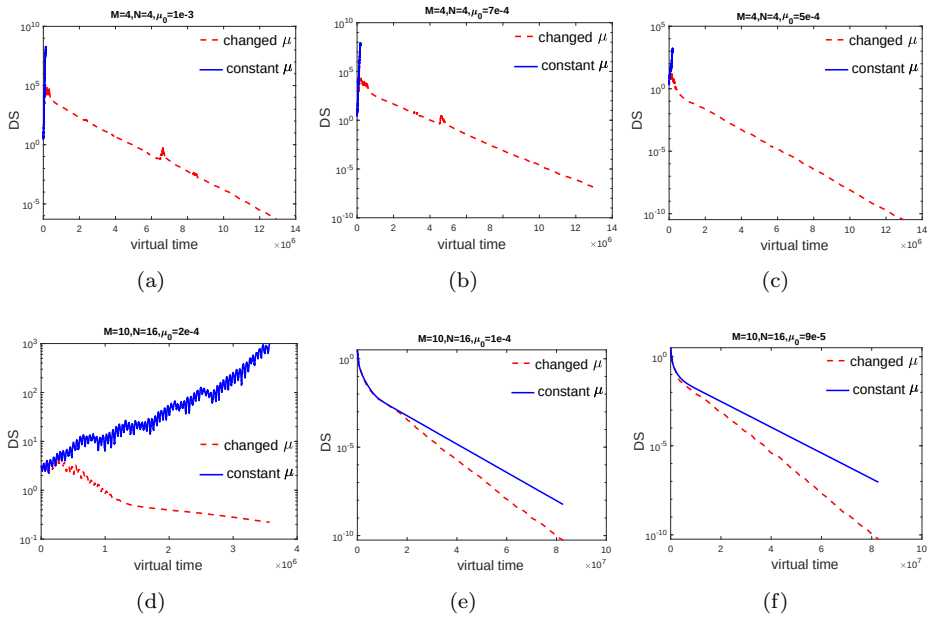


Figure 6.21: All parameters are the same with simulations in section 4.6. The automatic parameter tuning strategy can tune the step-length μ to a proper range and accelerate the original ABSGD-Ver2. This simulation adopts double-precision data type and thus DS achieves 10^{-10}.

BSGD and achieve faster reconstruction speed. In this section, the ABSGD is applied in a realistic parallel network by using non-blocking communications provided by mpi4py. In the Python environment, this means that we can use "Isend" and "Irecv" to perform the data communication in line 6,10 of Algo.6.5 and line 6,15 of Algo.6.6. For the code simplicity, however, this realistic application is developed at an initial stage, where only two parallel processes are created to control two parallel nodes. As a result, only ABSGD-Ver2, which only enable one parallel node to communicate with the master node, is compared with synchronous BSGD.

In simulations, ABSGD-Ver2 and synchronous BSGD are used to reconstruct a 3D cone beam scanning data with the usage of TIGRE toolbox. The scanning geometry is shown in Table.6.2. A 2-GPU workstation is used in this section and the ASTRA

Table 6.2: 3D cone-beam scanning geometry in ABSGD application

K in Fig.3.5	Detector size	Image size	Detector number
256	150×150	$64 \times 64 \times 64$	$8 \times K$
Voxel number	Projection views	OP in Fig.3.5	OD in Fig.3.5
$K \times K \times K$	$8 \times K$	1000	1000

toolbox can separately control the GPU to perform different projections. The GPUs are both Geforce GTX 2080Ti, they are named as GPU_0 and GPU_1. Two parallel processes are created and called $rank_0$ and $rank_1$. Each process, equipped with one

CPU, separately controls a single GPU to perform the FP and BP. The $rank_0$ process is also used as the master node, which is responsible for the update of $\mathbf{r}, \mathbf{g}, \mathbf{x}$. During data assignment, $rank_0$ sends data blocks using "Isend" while $rank_1$ receives data blocks using "Recv" (There is no need to use $Irecv$ because $rank_1$ will not work until the data is fully received). At the data gathering process, $rank_1$ sends a data block using "Send" (Again, there is no need to use "Isend" since $rank_1$ does not have computation task before next data block is received), while $rank_0$ receives the computed result using "Irecv". It can be seen that $rank_0$ does not have to wait during the data transformation stage. As a result, in ABSGD-Ver2, $rank_0$ can perform more computations than $rank_1$, whilst in synchronous BSGD, $rank_0$ has to wait for $rank_1$ until the data block finishes communication. To simulate the case when two GPUs have different computational abilities, a sleep function $time.sleep(ExtraDelay)$ is added after GPU_1 has performed its projections, in other words, GPU_1 is slower than GPU_0 by $ExtraDelay$ seconds. For different $ExtraDelay$ the SNR of the reconstructed $\mathbf{x}$ obtained by BSGD and ABSGD-Ver2 after 600s of wall time and the projection times for each process are shown in Fig.6.22. This illustrates that ABSGD-Ver2 allows the faster GPU_0 to perform more

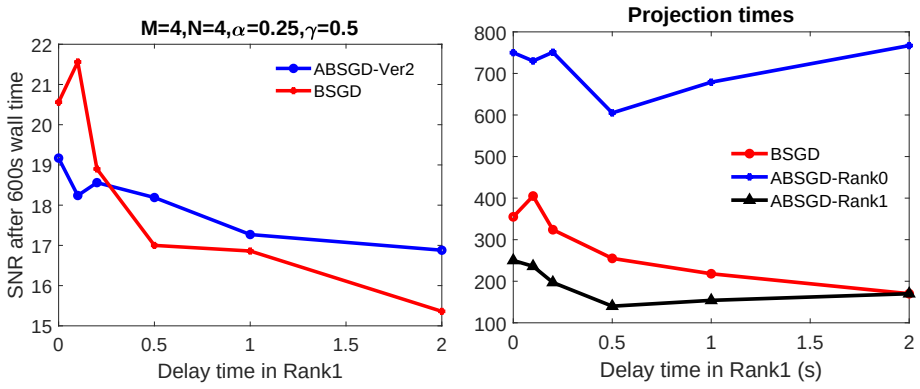


Figure 6.22: The step lengths μ in both methods have been carefully tuned for the fastest performance. When $Extradelay = 0$, which means that the two computation nodes have the same computation abilities, the BSGD is the best option. When $Extradelay$ increases, the ABSGD gradually shows better performance than BSGD. In the original case, the FP and BP both takes about 0.3 to 0.4s. When $Extradelay$ reaches 2s, it means that GPU_0 is 5 times faster than GPU_1.

projections even when $ExtraDelay = 0$. This is because the "Isend","Irecv" function allows $rank_0$ node to continue the work instead of waiting for $rank_1$ to finish data communication. When $ExtraDelay$ increases, since $rank_0$ always has to wait for $rank_1$ in BSGD, the reconstruction speed gradually slows down, whilst in ABSGD-Ver2, $rank_0$ can still frequently perform projections as well as the relevant update procedure, thus having a faster reconstruction speed than BSGD.

6.5 Conclusions

This chapter is a bit different from the previous chapters. Here the algorithm's property is not the main focus. Instead, realistic parallel applications of BSGD/CSGD and a flexible changes to BSGD were studied. In section 6.2, the general data communication scheme of CSGD and BSGD were proposed in Fig.6.7 and Fig.6.8. This illustrates that CSGD has a simpler communication scheme than BSGD. When the number of selected row and column blocks are pre-determined as αM and γN respectively, and p gradually increases from 1 to $\alpha\gamma MN$, then CSGD shows better scalability than BSGD, as shown in Fig.6.9. Furthermore, the actual reconstruction speed referenced by wall time is also compared between the two proposed algorithms. In simulations, the number of parallel nodes is gradually increased and α, γ are changed according to the different p. Simulations show that although BSGD has worse scalability than CSGD, BSGD still has faster reconstruction speed compared to CSGD, as shown in Fig.6.10. This is because BSGD has a faster reconstruction speed over " the usage of $\mathbf{A}_I^J$" than CSGD, and this property compensates BSGD's drawback in the inefficient communication scheme.

When applying BSGD and CSGD in a parallel network, the default communication scheme is synchronous. All parallel nodes will wait for each other until the slowest node has finished computation. After that they are allowed to communicate with the master node to send the computation results back to update $\mathbf{r}$ or $\mathbf{g}, \mathbf{x}$. If the parallel nodes have the same computation speed and can access the master node at the same time, this communication scheme works well. However in realistic parallel networks, it is possible that the master nodes have to communicate with each parallel computation node in a cyclic point-to-point way and the computation node may have different computation abilities. Furthermore, the communication network may also encounter unforeseeable delays during the reconstruction process. As a result, developing a more flexible communication scheme for BSGD is of interest in section 6.4. The proposed flexible communication enables the fast computation node to immediately communicate with the master node and does not have to wait for the slowest node. This communication is no longer synchronous but is asynchronous and thus is named as ABSGD. There are two versions of ABSGD: ABSGD-Ver1 and ABSGD-Ver2. In the ABSGD-Ver1, if there are several computation nodes that have finished the computation and are waiting for the communication with the master node, the master node will communicate with all of them. In the ABSGD-Ver2, the master node only communicates with one parallel node each time. Simulations have both verified that these two algorithms can reduce the waiting time of those fast computation nodes, enabling the faster computation nodes to perform more projections as well as iterations on $\mathbf{r}, \mathbf{g}, \mathbf{x}$, as shown in Fig.6.18, Fig.6.19 and Fig.6.20. This efficient communication scheme enables the ABSGD-Ver1 and ABSGD-Ver2 to both outperform the original synchronous BSGD when the computation nodes have different computation abilities, as shown in Fig.6.17. The ABSGD-Ver2 is applied in a 3D realistic reconstruction case where two parallel GPUs are used. Simulations

show that when the 2 GPUs are of the same computation abilities, the ABSGD has a slower reconstruction speed than BSGD. However, when one of the GPUs is slower than the another, the fast GPU can perform more projections instead of waiting for the slow GPU and thus the advantage of ABSGD appears, as shown in Fig.6.22.

There are two open areas of this chapter: 1) The experiments are mainly performed in a small scale problem to reduce the experiment time. The application of BSGD and CSGD on large case 3D cone beam dataset has not performed yet. In the large scale application, since each node is only equipped with four GPUs, it is possible to require data flow to be communicated between nodes. Such particular communication scheme and its property is not researched yet. 2) The programming of ABSGD in MPI environment is not fully finished. Currently only two nodes can be used for executing ABSGD. When using more nodes, a judgement criteria for deciding which node to communicate with the master node should be researched in detail.

Chapter 7

Conclusions

In this thesis, CT scanning is approximated as a linear system and the reconstruction process is regarded as an optimization of a quadratic objective function added with a regularization term. Current row and column action methods either require the reconstructed image vector $\mathbf{x}$ to be updated as a whole in each parallel computation node or require the computation nodes to have full access to the projection data $\mathbf{y}$. For large-scale CT reconstruction, this requirement causes multiple data communication between computation nodes (the node responsible for projection computation) and the master node (the node which stores all variables for iterative algorithms), making the communication overhead huge. In this thesis, two algorithms, CSGD and BSGD are proposed, aiming to reduce times of communication and thus to increase the reconstruction speed.

The contributions presented in this thesis can be concluded as three main parts. In the first two contributions, two parallel algorithms CSGD/BSGD are proposed respectively and their mathematical properties are studied in Chapters 3, 4. To be more specific, two algorithms are proposed and compared with each other as well as with other existing SIRT-type methods. All nodes were assumed to have the same computational abilities and they always communicate with the master node at the same time. The data communication and the following projection calculations are the most time-consuming part among the whole reconstruction process for all SIRT-type methods. As a result, the "usage of $\mathbf{A}_I^J$", which means an intact "request data block→projections→sends results back to the master node" circle for each computation node, is used as a time reference to reflect the speed of reconstruction. The first two contributions contained in these two chapters include the following contributions:

- CSGD is the first algorithm to be proposed and it is inspired by the steepest gradient descent algorithm. In Chapter 3, simulations have experimentally illustrated that it computes high-accuracy solutions regardless of the partitions of the system matrix $\mathbf{A}$. This method reduces the computation cost in each iteration and also enables a more flexible parallel application, at the cost of introducing a more

stochastic update direction. Besides, the computation efficiency is much higher than block ADMM, which is another main reference method in this thesis apart from the first order methods. Simulations have verified the speed advantage of CSGD when compared with block ADMM.

- BSGD is inspired by the SAG. It uses a more complicated data communication scheme than CSGD. This is caused by the accumulation of the previous sub-gradient ingredients $\hat{\mathbf{g}}_J^i$, thus making the error variance of the estimated gradient decrease along with iterations. The complicated communication scheme requires 2 inner-loops to update $\mathbf{r}$, $\mathbf{g}$ and $\mathbf{x}$ separately, which is the same as the other SIRT-type row action methods. Despite the same communication scheme, BSGD does not have to iterate over all of $\mathbf{x}$ each time and it computes to high-accuracy solutions even if the percentage of selected column blocks $\gamma < 1$. It is similar to CSGD, in that it allows a flexible sampling of sub-matrices $\mathbf{A}_I^J$ and converges regardless of the shape of $\mathbf{A}_I^J$ or the selection criteria on α, γ. In Chapter 4, BSGD uses the same sampling method as CSGD. This can simplify the data communication by minimizing $\alpha = \frac{1}{M}$ and adjusting γ according to the available number of computation nodes, enabling $\mathbf{x}$ blocks to be independently updated within the parallel computation nodes. Importance sampling is also applicable to BSGD and simulations have verified that BSGD-IM has the fastest reconstruction speed compared with the other existing SIRT-type methods including mini-batch SGD, SAG, SVRG.

The third contribution is the further explorations on two developed algorithms. The exploration includes : 1) the importance sampling trick and automatic parameter tuning trick. The discussions have presented during the introduction of CSGD and BSGD during Chapter 3 and 4. 2) The application of two proposed algorithms in few-view projection data case. Simulation results have shown that they can substitute the GD in the proximal methods to approximately optimize a TV-regularized objective function. 3) The application of CSGD and BSGD in realistic parallel network is performed in Iridis supercomputer, which demonstrates two algorithms scalability. Furthermore, an asynchronous application framework of BSGD is developed to further increase the reconstruction speed. The third contribution is expanded as the following:

- An importance sampling method is proposed and is available for two proposed methods. In the original CSGD and BSGD, the image is divided into N sub-images and projections of each sub-image are not distributed to the whole detector but on a limited sub-detector area. The importance sampling method divides the detector into many sub-detectors and only samples partial of sub-detectors. This enables each row block to contain more projection views and let sub-detectors with massive non-zero projection data be more frequently used in iterations than those

with loads of zero projection data. Simulations have experimentally verified the improvement brought by importance sampling.

- In realistic settings, the step-length μ may be set too small which leads to a slow reconstruction speed. An automatic parameter tuning strategy is thus proposed to accelerate the case when initial step-length is set improperly small. Furthermore, the Katyusha acceleration trick is adopted for BSGD. Instead of applying a momentum step after every single stochastic iteration, the BSGD-KatyushaX^s applies a momentum acceleration under an update frequency f. Simulations show that when $f \geq 2M$, the BSGD-KatyushaX^s effectively accelerates BSGD and BSGD-IM even when they are well tuned to the fastest reconstruction speeds.

- Simulations in Chapter 3 and 4 mainly discuss two algorithms' performance when there is enough projection data. When using few projections, only applying the SIRT-type algorithms to optimize the quadratic objective function cannot obtain a high-accuracy solution and the reconstructed image shows severe artefacts. A popular method to increase the reconstructed image quality is to use a TV de-noising procedure after the gradient-descent movement. Simulations in Chapter 5 prove that CSGD and BSGD can be combined with TV de-noising. The combination is flexible. It can perform CSGD/BSGD iterations with predefined frequencies, followed by a TV de-noising on the whole of $\mathbf{x}$, or on γN image blocks. Under two different combinations, the frequencies of CSGD/BSGD iterations are different to ease the comparison with the other SIRT-type methods with TV constraints. Simulations have verified that partial de-noising on γN blocks, called BSGD/CSGD-TV(partial), reconstructs the scanned object with high-SNR without sacrificing reconstruction speed and image quality compared with BSGD/CSGD-TV(whole), which perform TV de-noising on the whole of $\mathbf{x}$. Both methods can outperform the other existing SIRT-type methods due to the reduced communication overhead.

- The scalability of BSGD and CSGD are compared with each other and simulations are performed in a Python environment. Each node has the same computation speed and they execute code simultaneously. CSGD is shown to have better scalability than BSGD due to a smaller communication overhead and reduced communication times. However, BSGD is faster than CSGD in realistic parallel computation environment. As a result, in the following discussions on asynchronous applications, only BSGD is discussed.

- In realistic parallel networks, there may be a time delay for each node to receieve the data from the master node because the master node adopts a cyclic point-to-point communication method or the communication network has unforeseeable delay during the reconstruction process. As a result, developing a more flexible communication scheme of BSGD is of interest in the last part of the thesis. The flexible communication increases each parallel node's independence and enables

them to communicate with the master node asynchronously. It is called ABSGD. There are two versions of ABSGD: ABSGD-Ver1 and ABSGD-Ver2. In the ABSGD-Ver1, if there are several computation nodes that have finished the computation and are waiting for the communication with the master node, the master node will communicate with all of them. In the ABSGD-Ver2, the master node only communicates with one parallel node each time. Simulations have verified that these two algorithms can reduce the waiting time of those fast computation nodes, enabling the fast computation nodes to perform more projections as well as iterations on $\mathbf{r}, \mathbf{g}, \mathbf{x}$, outperforming the original BSGD.

7.1 Future work

Both CSGD and BSGD are inspired by existing SIRT-type methods and can be viewed as improvements on the steepest gradient descent and SAG methods respectively. Since both algorithms uses only a block of projection data and image, the update direction is rather stochastic and cannot be viewed as unbiased estimation of the gradient direction. This property makes the mathematical analysis rather difficult. Currently only the fixed point of two algorithms are performed. The fixed point of BSGD is located at the least square solution whilst the CSGD's fixed point is located at a weighted least square solution. From this result, a conjecture is proposed that the BSGD converges to a solution that is much closer to CSGD. Simulation results have shown corresponding results. However, rigorous convergence analysis proving whether the BSGD converges to the least square solution on expectation and whether the CSGD converges to a least square solution on expectation is missing in this thesis. Due to the lack of theoretical proof, from the Chapter 3 to 5, the advantages of CSGD/BSGD over the other methods are both experimentally illustrated instead of mathematically. A more rigorous proof on the advantage of BSGD/CSGD over other SIRT-type algorithms needs to be performed.

Simulation results for noise-free and slight noise dataset have shown that the CSGD and BSGD can obtain incremental SNR trends, which hints that they have the ability to reduce the data fidelity. This means that the CSGD and BSGD can replace the gradient descent procedure and can combine with TV-based denoising procedure to approximately optimize TV-based optimization, which is of more realistic application value. In fact, when the dataset is contaminated by high level noise, situations have shown that with a proper early termination strategy, for example, terminating the iteration when the SNR has achieved its peak value, the CSGD and BSGD show similar relative reconstruction speed comparison results under different α, γ parameter settings (for example, shown in Fig.3.9-3.12, Fig.4.1-4.3), and the simulations also prove the claim made in Eq.3.8 for how to determine α, γ for a fast reconstruction speed. However, after introducing the early stopping strategy, it then becomes difficult to convince readers that the CSGD and BSGD are converging. Furthermore, in simulations, when

to determine the CSGD and BSGD is easy to determine, as the SNR trend is visible (the $\mathbf{x}_{true}$ is a known vector), but in realistic applications when to stop the iteration is difficult to determine and it becomes another research topic. Consider the fact that the main focuses of Chapter 3 and 4 are to illustrate that the CSGD and BSGD has the ability to reduce the data fidelity and obtain an increment SNR trend and to show the recommended parameter(mainly α, γ) setting, this thesis does not present the simulations under severe noise where the semi-convergence phenomena is significant. The detailed discussions on semi-convergence phenomena is an open area.

Furthermore, as this thesis mainly focuses on the two algorithms' basic properties, the performance of the methods is mainly illustrated on small scale CT scanning problems to save time. It thus lacks more realistic applications to large scale CT dataset. In future work, more CSGD and BSGD applications on realistic datasets are needed to be performed. In the asynchronous application of BSGD, the code currently only covers the 2-GPU case, where $rank_0$ and $rank_1$ communicate with each other using "Isend,Irecv". A more general code that enables more than 2 GPUs to get involved in the reconstruction is missing in this thesis.

Appendix A

Open area for mathematical analysis of CSGD and BSGD

This appendix includes the mathematical analysis conducted on both BSGD and CSGD. Fixed point analysis has performed to find a solution that makes the iteration generate a stable iteration results sequence.

A.1 CSGD fixed point analysis

Whilst a formal convergence proof of general CSGD is not available yet, it is instructive to analyse the fixed points of the algorithm. The deterministic version of the algorithm with $\alpha = \gamma = 1$ is first analysed.

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

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

and

$$\begin{aligned}\mathbf{z}_I^{k+1} &= \sum_{j=1}^{N} \mathbf{z}_I^{j\,k} = \sum_{j=1}^{N} \mathbf{A}_I^{J_j}(\mathbf{x}_{J_j}^k + \mu_j^i \mathbf{g}_{J_j}^i) \\ &= \sum_{j=1}^{N} \mathbf{A}_I^{J_j}(\mathbf{x}_{J_j}^k + \mu_j^i (\mathbf{A}_I^{J_j})^T \mathbf{r}_I^k) \\ &= \mathbf{A}_I \mathbf{x}^k + \sum_{j=1}^{N} \mu_j^i \mathbf{A}_I^{J_j}(\mathbf{A}_I^{J_j})^T(\mathbf{y}_I^k - \mathbf{z}_I^k) \\ &= \mathbf{A}_I \mathbf{x}^k + \mathbf{S}_I(\mathbf{y}_I^k - \mathbf{z}_I^k),\end{aligned} \tag{A.2}$$

where $\mathbf{S}_I$ is a determined matrix and is defined as $\mathbf{S}_I = \sum_{j=1}^{N} \mu_j^i \mathbf{A}_I^{J_j}(\mathbf{A}_I^{J_j})^T$.

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

$$\mathbf{z}_I^\star = (I + \mathbf{S}_I)^{-1}\mathbf{A}_I \mathbf{x}^\star + (I + \mathbf{S}_I)^{-1}\mathbf{S}_I \mathbf{y}_I \tag{A.3}$$

and

$$\sum_{i=1}^{M} \mu_j^i (\mathbf{A}_{I_i}^J)^T(\mathbf{y}_{I_i} - \mathbf{z}_{I_i}^\star) = \mathbf{0}. \tag{A.4}$$

Eq.A.3 can be expanded

$$\mathbf{z}^\star = (I + \mathbf{S}_T)^{-1}\mathbf{A}\mathbf{x}^\star + (I + \mathbf{S}_T)^{-1}\mathbf{S}_T \mathbf{y}, \tag{A.5}$$

where $\mathbf{S}_T$ is a block diagonal matrix:

$$\mathbf{S}_T = \begin{bmatrix} \mathbf{S}_{I_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{I_2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S}_{I_M} \end{bmatrix}. \tag{A.6}$$

Note that $(I + \mathbf{S}_T)$ is a positive semi-definite matrix if μ_j^i are positive. Define a diagonal matrix $\mathbf{D}_j$ with diagonal entries μ_j^i where $\mathbf{D}_j(h,h) = \mu_j^i$ if $h \in I_i$. Eq.A.4 can thus be written as

$$(\mathbf{D}_j \mathbf{A}^J)^T(\mathbf{y} - \mathbf{z}^\star) = \mathbf{0}. \tag{A.7}$$

Combining Eq.A.5 and Eq.A.7 gives

$$\begin{aligned}\mathbf{0} &= (\mathbf{D}_j\mathbf{A}^J)^T(\mathbf{y} - (I+\mathbf{S}_T)^{-1}\mathbf{A}\mathbf{x}^\star - (I+\mathbf{S}_T)^{-1}\mathbf{S}_T\mathbf{y}) \\ &= (\mathbf{D}_j\mathbf{A}^J)^T(I+\mathbf{S}_T)^{-1}(\mathbf{y}-\mathbf{A}\mathbf{x}^\star).\end{aligned} \tag{A.8}$$

This equation has to hold for all J. As $\mathbf{D}_j$ is a diagonal matrix, this implies that

$$\mathbf{0} = \mathbf{A}^T(I+\mathbf{S}_T)^{-1}(\mathbf{y}-\mathbf{A}\mathbf{x}^\star), \tag{A.9}$$

which shows

$$\mathbf{x}^\star = \left(\mathbf{A}^T(I+\mathbf{S}_T)^{-1}\mathbf{A}\right)^{-1}\mathbf{A}^T(I+\mathbf{S}_T)^{-1}\mathbf{y} \tag{A.10}$$

is a weighted least squares solution.

A.2 BSGD fixed point analysis

In this section, some mathematical deductions are presented to show that the fixed point of BSGD locates at the least square solution. To simplify the deduction, the sampling of I_i, J_j in inner loops are considered as independent to each other.

Notation: Let us vectorise the sets $\{(\mathbf{z}^j_{I_i})^k\}$ and $\{(\hat{\mathbf{g}}^i_{J_j})^k\}$ and put their elements into the vectors $\tilde{\mathbf{z}}^k$ and $\tilde{\mathbf{g}}^k$. Let $\overline{\mathbf{A}}$ be defined as

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A}^{J_1}_{I_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}^{J_1}_{I_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & & & \vdots \\ \mathbf{A}^{J_1}_{I_M} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{J_2}_{I_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{J_2}_{I_2} & \mathbf{0} & \mathbf{0} \\ \vdots & & & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}^{J_N}_{I_M} \end{bmatrix} \in \mathbb{R}^{Nr*c}$$

and let

$$\overline{\mathbf{A}^T} = \begin{bmatrix} (\mathbf{A}^{J_1}_{I_1})^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ (\mathbf{A}^{J_2}_{I_1})^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & & & \vdots \\ (\mathbf{A}^{J_N}_{I_1})^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathbf{A}^{J_1}_{I_2})^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathbf{A}^{J_2}_{I_2})^T & \mathbf{0} & \mathbf{0} \\ \vdots & & & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & (\mathbf{A}^{J_N}_{I_M})^T \end{bmatrix} \in \mathbb{R}^{Mc*r}.$$

We have:

$$\mathbf{x} \in \mathbb{R}^c$$

$$\mathbf{y} \in \mathbb{R}^r$$

$$\tilde{\mathbf{z}} \in \mathbb{R}^{Nr}$$

$$\tilde{\mathbf{g}} \in \mathbb{R}^{Mc}.$$

Let us also introduce the matrix $\bar{\mathbf{I}}_{Mc} = \big[\mathbf{I}_c, \mathbf{I}_c, \cdots \mathbf{I}_c\big]$, where we concatenate M identity matrices $\mathbf{I}_c$ each of size $c * c$ and define $\bar{\mathbf{I}}_{Nr}$ similarly.

With this notation $\bar{\mathbf{I}}_{Nr}\overline{\mathbf{A}} = \mathbf{A}$ and $\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T} = \mathbf{A}^T$.

Since

$$\mathbf{z}_{I_i}^j = \mathbf{A}_{I_i}^{J_j}\mathbf{x}_{J_j}$$

and

$$\hat{\mathbf{g}}_{J_j}^i = (\mathbf{A}_{I_i}^{J_j})^T(\mathbf{y}_{I_i} - \sum_j \mathbf{z}_{I_i}^j).$$

then

$$\sum_j \begin{bmatrix} (\mathbf{z}_{I_1}^j)^{k+1} \\ (\mathbf{z}_{I_2}^j)^{k+1} \\ \vdots \\ (\mathbf{z}_{I_M}^j)^{k+1} \end{bmatrix} = \bar{\mathbf{I}}_{Nr}\tilde{\mathbf{z}}^{k+1}$$

and

$$\sum_i \begin{bmatrix} (\hat{\mathbf{g}}_{J_1}^i)^{k+1} \\ (\hat{\mathbf{g}}_{J_2}^i)^{k+1} \\ \vdots \\ (\hat{\mathbf{g}}_{J_N}^i)^{k+1} \end{bmatrix} = \bar{\mathbf{I}}_{Mc}\tilde{\mathbf{g}}^{k+1}.$$

To encode the random updates over subsets of index pairs $\{i, j\}$, random matrix whose elements are random numbers are introduced. Here introduce three random diagonal matrices $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$, which are diagonal matrices whose diagonal entries are either 0 or 1. Note that $\mathbf{R}_1$ is a block matrix with MN square blocks of size r/M along the diagonal, where random $\alpha\gamma MN$ blocks among total MN blocks (defined as $\binom{MN}{\alpha\gamma MN}$) are identity matrices with the remaining blocks being 0. Similarly $\mathbf{R}_2$ and $\mathbf{R}_3$ are block diagonal with MN and N square blocks of size c/N along the diagonal. For $\mathbf{R}_2$ there are $\binom{MN}{\alpha\gamma MN}$ non-zero blocks along the diagonal, whilst for $\mathbf{R}_3$ we have n_3 blocks, where

n_3 is the number of distinct indices j drawn. We have

$$\mathbf{R}_1 = \begin{bmatrix} \delta^1_{1,1}\mathbf{I}_{r/M} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \delta^1_{1,2}\mathbf{I}_{r/M} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ & & \ddots & & & \\ \mathbf{0} & \mathbf{0} & \cdots & \delta^1_{i,j}\mathbf{I}_{r/M} & \cdots & \mathbf{0} \\ & & & & \ddots & \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \delta^1_{M,N}\mathbf{I}_{r/M} \end{bmatrix} \in \mathbb{R}^{Nr*Nr}$$

,

$$\mathbf{R}_2 = \begin{bmatrix} \delta^2_{1,1}\mathbf{I}_{c/N} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \delta^2_{1,2}\mathbf{I}_{c/N} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ & & \ddots & & & \\ \mathbf{0} & \mathbf{0} & \cdots & \delta^2_{i,j}\mathbf{I}_{c/N} & \cdots & \mathbf{0} \\ & & & & \ddots & \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \delta^2_{M,N}\mathbf{I}_{c/N} \end{bmatrix} \in \mathbb{R}^{Mc*Mc}$$

,

$$\mathbf{R}_3 = \begin{bmatrix} \delta^3_{1}\mathbf{I}_{c/N} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \delta^3_{2}\mathbf{I}_{c/N} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ & & \ddots & & & \\ \mathbf{0} & \mathbf{0} & \cdots & \delta^3_{j}\mathbf{I}_{c/N} & \cdots & \mathbf{0} \\ & & & & \ddots & \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \delta^3_{N}\mathbf{I}_{c/N} \end{bmatrix} \in \mathbb{R}^{c*c}$$

, where $\delta^1_{i,j}$, $\delta^2_{i,j}$ and δ^3_j are random coefficients whose values are 0 or 1, which reflects which $\mathbf{A}^{J_j}_{I_i}$ and $\mathbf{x}_{J_j}$ are selected.

We thus have $\tilde{\mathbf{z}}^{k+1} = \tilde{\mathbf{z}}^k + \mathbf{R}_1(\overline{\mathbf{A}}\mathbf{x}^k - \tilde{\mathbf{z}}^k)$, $\tilde{\mathbf{g}}^{k+1} = \tilde{\mathbf{g}}^k + \mathbf{R}_2(\overline{\mathbf{A}^T}\mathbf{r}^{k+1} - \tilde{\mathbf{g}}^k)$ and $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{R}_3\mu\bar{\mathbf{I}}_{Mc}\tilde{\mathbf{g}}^{k+1}$

With this notation, we can thus write:

$$\mathbf{r}^k = \mathbf{y} - \bar{\mathbf{I}}_{Nr}\tilde{\mathbf{z}}^k \tag{A.11}$$

so that the update of $\tilde{\mathbf{z}}$, $\tilde{\mathbf{g}}$ and $\mathbf{x}$ becomes:

$$\tilde{\mathbf{z}}^{k+1} = \tilde{\mathbf{z}}^k + \mathbf{R}_1\left[\overline{\mathbf{A}}\mathbf{x}^k - \tilde{\mathbf{z}}^k\right], \tag{A.12}$$

$$\tilde{\mathbf{g}}^{k+1} = \tilde{\mathbf{g}}^k + \mathbf{R}_2\left[\overline{\mathbf{A}^T}\left(\mathbf{y} - \bar{\mathbf{I}}_{Nr}\tilde{\mathbf{z}}^{k+1}\right) - \tilde{\mathbf{g}}^k\right] \tag{A.13}$$

and

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{R}_3 \left(\mu \bar{\mathbf{I}}_{Mc} \tilde{\mathbf{g}}^{k+1} \right). \tag{A.14}$$

Inserting the first equation into the second and then the second equation into the third, we get the following recursions:

$$\tilde{\mathbf{z}}^{k+1} = \tilde{\mathbf{z}}^k + \mathbf{R}_1 \left[\overline{\mathbf{A}} \mathbf{x}^k - \tilde{\mathbf{z}}^k \right], \tag{A.15}$$

i.e .

$$\tilde{\mathbf{z}}^{k+1} = \left(\mathbf{I}_{Nr} - \mathbf{R}_1 \right) \tilde{\mathbf{z}}^k + \mathbf{R}_1 \overline{\mathbf{A}} \mathbf{x}^k, \tag{A.16}$$

$$\tilde{\mathbf{g}}^{k+1} = \tilde{\mathbf{g}}^k + \mathbf{R}_2 \left[\overline{\mathbf{A}^T} \left(\mathbf{y} - \left(\bar{\mathbf{I}}_{Nr} \left(\mathbf{I}_{Nr} - \mathbf{R}_1 \right) \tilde{\mathbf{z}}^k + \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \mathbf{x}^k \right) \right) - \tilde{\mathbf{g}}^k \right] \tag{A.17}$$

i.e .

$$\tilde{\mathbf{g}}^{k+1} = \left(\mathbf{I}_{Mc} - \mathbf{R}_2 \right) \tilde{\mathbf{g}}^k + \mathbf{R}_2 \overline{\mathbf{A}^T} \mathbf{y} - \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} (\mathbf{I}_{Nr} - \mathbf{R}_1) \tilde{\mathbf{z}}^k - \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \mathbf{x}^k \tag{A.18}$$

and

$$\begin{aligned} \mathbf{x}^{k+1} \;=\; & \mathbf{x}^k + \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \left(\mathbf{I}_{Mc} - \mathbf{R}_2 \right) \tilde{\mathbf{g}}^k + \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \mathbf{y} \\ & - \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} (\mathbf{I}_{Nr} - \mathbf{R}_1) \tilde{\mathbf{z}}^k - \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \mathbf{x}^k. \end{aligned} \tag{A.19}$$

i.e.

$$\begin{aligned} \mathbf{x}^{k+1} \;=\; & \left(\mathbf{I}_c - \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \right) \mathbf{x}^k \\ & + \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \left(\mathbf{I}_{Mc} - \mathbf{R}_2 \right) \tilde{\mathbf{g}}^k \\ & + \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \mathbf{y} \\ & - \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} (\mathbf{I}_{Nr} - \mathbf{R}_1) \tilde{\mathbf{z}}^k. \end{aligned}$$

or in matrix form:

$$\begin{bmatrix} \tilde{\mathbf{z}}^{k+1} \\ \tilde{\mathbf{g}}^{k+1} \\ \mathbf{x}^{k+1} \end{bmatrix} = \mathbf{M}_k \begin{bmatrix} \tilde{\mathbf{z}}^k \\ \tilde{\mathbf{g}}^k \\ \mathbf{x}^k \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{R}_2 \overline{\mathbf{A}^T} \mathbf{y} \\ \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \mathbf{y} \end{bmatrix} \tag{A.20}$$

where

$$\mathbf{M}_k = \begin{bmatrix} \left(\mathbf{I}_{Nr} - \mathbf{R}_1 \right) & \mathbf{0} & \mathbf{R}_1 \overline{\mathbf{A}} \\ -\mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} (\mathbf{I}_{Nr} - \mathbf{R}_1) & \left(\mathbf{I}_{Mc} - \mathbf{R}_2 \right) & -\mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \\ -\mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} (\mathbf{I}_{Nr} - \mathbf{R}_1) & \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \left(\mathbf{I}_{Mc} - \mathbf{R}_2 \right) & \left(\mathbf{I}_c - \mu \mathbf{R}_3 \bar{\mathbf{I}}_{Mc} \mathbf{R}_2 \overline{\mathbf{A}^T} \bar{\mathbf{I}}_{Nr} \mathbf{R}_1 \overline{\mathbf{A}} \right) \end{bmatrix}$$

Theorem A.1. *For all* $\mathbf{R}_1$, $\mathbf{R}_2$ *and* $\mathbf{R}_3$, *the vector*

$$\begin{bmatrix} \mathbf{z}^* \\ \mathbf{g}^* \\ \mathbf{x}^* \end{bmatrix}$$

is a fixed point of the algorithm iff

$$\overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*) = \mathbf{g}^*, \tag{A.21}$$

$$\mathbf{z}^* = \begin{bmatrix} \mathbf{A}_{I_1}^{J_1}\mathbf{x}_{J_1}^* \\ \mathbf{A}_{I_2}^{J_1}\mathbf{x}_{J_1}^* \\ \vdots \\ \mathbf{A}_{I_M}^{J_N}\mathbf{x}_{J_N}^* \end{bmatrix} = \overline{\mathbf{A}}\mathbf{x}^*$$

and

$$\bar{\mathbf{I}}_{Mc}\mathbf{g}^* = \bar{\mathbf{I}}_{Mc}\begin{bmatrix} \mathbf{A}_{I_1}^T(\mathbf{y}_{I_1} - \mathbf{A}_{I_1}\mathbf{x}^*) \\ \mathbf{A}_{I_2}^T(\mathbf{y}_{I_2} - \mathbf{A}_{I_2}\mathbf{x}^*) \\ \vdots \\ \mathbf{A}_{I_M}^T(\mathbf{y}_{I_M} - \mathbf{A}_{I_M}\mathbf{x}^*) \end{bmatrix} = \bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*) = \mathbf{0}.$$

Proof. To prove the theorem, we look at each line of the Eq.A.20.

The first line

$$\mathbf{z}^* - \mathbf{R}_1\mathbf{z}^* + \mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^* = \mathbf{z}^* \tag{A.22}$$

holds for all $\mathbf{R}_1$ iff $\mathbf{z}^* = \overline{\mathbf{A}}\mathbf{x}^*$. Thus, using $\mathbf{z}^* = \overline{\mathbf{A}}\mathbf{x}^*$, the second line gives

$$-\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}(\mathbf{I}_{Nr} - \mathbf{R}_1)\mathbf{z}^* + (\mathbf{I}_{Mc} - \mathbf{R}_2)\mathbf{g}^* - \mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^* + \mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{y} = \mathbf{g}^* \tag{A.23}$$

If it is expanded it can be seen that:

$$-\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\overline{\mathbf{A}}\mathbf{x}^* + \mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^* - \mathbf{R}_2\mathbf{g}^* - \mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^* + \mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{y} = \mathbf{0} \tag{A.24}$$

The above equation can be simplified as

$$\mathbf{g}^* = \overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*) \tag{A.25}$$

where we have used the equality $\bar{\mathbf{I}}_{Nr}\overline{\mathbf{A}} = \mathbf{A}$. Using $\mathbf{g}^* = \overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*)$ the last line then gives

$$\begin{aligned}
& \mathbf{x}^* + \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^* - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\overline{\mathbf{A}}\mathbf{x}^* && \text{(A.26)}\\
& +\mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{y} - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{y} - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^* + \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^*\\
& -\mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\mathbf{R}_1\overline{\mathbf{A}}\mathbf{x}^*\\
& +\mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{y}\\
= \;& \mathbf{x}^* - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\bar{\mathbf{I}}_{Nr}\overline{\mathbf{A}}\mathbf{x}^* && \text{(A.27)}\\
& +\mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{y} - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^* + \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^*\\
= \;& \mathbf{x}^* - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^* && \text{(A.28)}\\
& +\mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{y} - \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^* + \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\mathbf{R}_2\overline{\mathbf{A}^T}\mathbf{A}\mathbf{x}^*\\
= \;& \mathbf{x}^* + \mu\mathbf{R}_3\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*) = \mathbf{x}^*, && \text{(A.29)}
\end{aligned}$$

which can only hold for all $\mathbf{R}_3$ iff $\bar{\mathbf{I}}_{Mc}\overline{\mathbf{A}^T}(\mathbf{y} - \mathbf{A}\mathbf{x}^*) = \mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{x}^*) = \bar{\mathbf{I}}_{Mc}\mathbf{g}^* = \mathbf{0}$. This means that the fixed point of the BSGD is located at the least square solution. Despite that currently it cannot be proved that the BSGD can finally on expectation converges to the fixed point, at least the analysis hints that the BSGD can approach to the least square solution closer than CSGD and the previous simulations have verified this conjecture.

References

Allen-Zhu, Z. (2017). Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244.

Allen-Zhu, Z. (2018). Katyusha x: Simple momentum method for stochastic sum-of-nonconvex optimization. In *International Conference on Machine Learning*, pages 179–185.

Andersen, A. H. and Kak, A. C. (1984). Simultaneous algebraic reconstruction technique (sart): a superior implementation of the art algorithm. *Ultrasonic imaging*, 6(1):81–94.

Atilim Gunes, B., David Martinez, R., Mark, S., and Frank, W. (2017). Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*.

Bai, Z.-Z. and Wu, W.-T. (2018). On convergence rate of the randomized kaczmarz method. *Linear Algebra and Its Applications*, 553:252–269.

Banjak, H., Costin, M., Vienne, C., and Kaftandjian, V. (2016). X-ray computed tomography reconstruction on non-standard trajectories for robotized inspection. In *proceedings of the World Conference of NDT*.

Barker, B. (2015). Message passing interface (mpi). In *Workshop: High Performance Computing on Stampede*, volume 262.

Bartscher, M., Hilpert, U., Goebbels, J., and Weidemann, G. (2007). Enhancement and proof of accuracy of industrial computed tomography (ct) measurements. *CIRP Annals-Manufacturing Technology*, 56(1):495–498.

Bayford, R. H. and Lionheart, B. R. (2004). Biomedical applications of electrical impedance tomography. *Physiological Measurement*, 25(1).

Beck, A. and Teboulle, M. (2009a). Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems. *IEEE Transactions on Image Processing*, 18(11):2419–2434.

Beck, A. and Teboulle, M. (2009b). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202.

Beister, M., Kolditz, D., and Kalender, W. A. (2012). Iterative reconstruction methods in x-ray ct. *Physica medica*, 28(2):94–108.

Belge, M., Kilmer, M. E., and Miller, E. L. (2000). Wavelet domain image restoration with adaptive edge-preserving regularization. *IEEE Transactions on Image Processing*, 9(4):597–608.

Ben-Israel, A. and Greville, T. N. (2003). *Generalized inverses: theory and applications*, volume 15. Springer Science & Business Media.

Benson, T. M., De Man, B. K., Fu, L., and Thibault, J.-B. (2010). Block-based iterative coordinate descent. In *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*, pages 2856–2859. IEEE.

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

Biguri, A., Dosanjh, M., Hancock, S., and Soleimani, M. (2016). Tigre: a matlab-gpu toolbox for cbct image reconstruction. *Biomedical Physics & Engineering Express*, 2(5):055010.

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

Bilbao-Castro, J., Carazo, J., Garcia, I., and Fernández, J. (2006). Parallelization of reconstruction algorithms in three-dimensional electron microscopy. *Applied mathematical modelling*, 30(8):688–701.

Blatt, D., Hero, A. O., and Gauchman, H. (2007). A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51.

Blumensath, T. and Davies, M. E. (2009). Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 27(3):265–274.

Borsic, A., McLeod, C., and Lionheart, W. (2001). Total variation regularisation in eit reconstruction. In *Proceedings of the 2nd World Congress on Industrial Process Tomography*, volume 433.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

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

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122.

Bressert, E. (2012). *SciPy and NumPy: an overview for developers*. O'Reilly Media, Inc.

Calvetti, D., Morigi, S., Reichel, L., and Sgallari, F. (2000). Tikhonov regularization and the l-curve for large discrete ill-posed problems. *Journal of computational and applied mathematics*, 123(1-2):423–446.

Cand, E. J. and Wakin, M. B. (2008). An introduction to compressive sampling. *IEEE signal processing magazine*, 25(2):21–30.

Ceder, V. (2010). *The quick Python book*. Manning Publications Co.

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

Censor, Y. (1983). Finite series-expansion reconstruction methods. *Proceedings of the IEEE*, 71(3):409–419.

Censor, Y., Eggermont, P. P., and Gordon, D. (1983). Strong underrelaxation in kaczmarz's method for inconsistent systems. *Numerische Mathematik*, 41(1):83–92.

Censor, Y., Gordon, D., and Gordon, R. (2001a). Bicav: A block-iterative parallel algorithm for sparse systems with pixel-related weighting. *IEEE Transactions on Medical Imaging*, 20(10):1050–1060.

Censor, Y., Gordon, D., and Gordon, R. (2001b). Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel computing*, 27(6):777–808.

Censor, Y. and Herman, G. T. (1987). On some optimization techniques in image reconstruction from projections. *Applied Numerical Mathematics*, 3(5):365–391.

Chang, K. W., Hsieh, C. J., and Lin, C. J. (2008). Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9(Jul):1369–1398.

Chen, J. and Gu, Q. (2016). Accelerated stochastic block coordinate gradient descent for sparsity constrained nonconvex optimization. In *UAI*.

Chen, J., Wu, M., and Yao, Y. (2014). Accelerating ct iterative reconstruction using admm and nesterov's methods.

Chen, Z., Jin, X., Li, L., and Wang, G. (2013). A limited-angle ct reconstruction method based on anisotropic tv minimization. *Physics in medicine and biology*, 58(7):2119–2141.

Chun, S. Y., Dewaraja, Y. K., and Fessler, J. A. (2014). Alternating direction method of multiplier for tomography with nonlocal regularizers. *IEEE transactions on medical imaging*, 33(10):1960–1968.

Combettes, P. L. and Pesquet, J. C. (2011). Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer.

Conghui, T., Shiqian, M., Yuhong, D., and Yuqiu, Q. (2016). Barzilai-borwein step size for stochastic gradient descent. *Advances in Neural Information Processing Systems*, pages 685–693.

Dai, L. and Schön, T. B. (2015). On the exponential convergence of the kaczmarz algorithm. *IEEE Signal Processing Letters*, 22(10):1571–1574.

De Chiffre, L., Carmignato, S., Kruth, J.-P., Schmitt, R., and Weckenmann, A. (2014). Industrial applications of computed tomography. *CIRP Annals-Manufacturing Technology*, 63(2):655–677.

De Man, B., Basu, S., Thibault, J.-B., Hsieh, J., Fessier, J., Bouman, C., and Sauer, K. (2005). A study of four minimization approaches for iterative reconstruction in x-ray ct. In *Nuclear Science Symposium Conference Record, 2005 IEEE*, volume 5, pages 2708–2710. IEEE.

Dilz, R. J., Schröder, L., Moriakov, N., Sonke, J.-J., and Teuwen, J. (2019). Learned sirt for cone beam computed tomography reconstruction. *arXiv preprint arXiv:1908.10715*.

Donatelli, M., Neuman, A., and Reichel, L. (2012). Square regularization matrices for large linear discrete ill-posed problems. *Numerical Linear Algebra with Applications*, 19(6):896–913.

Elfving, T., Hansen, P. C., and Nikazad, T. (2014). Semi-convergence properties of kaczmarz's method. *Inverse Problems*, 30(5):055007.

Elfving, T., Hansen, P. C., and Nikazad, T. (2017). Convergence analysis for column-action methods in image reconstruction. *Numerical Algorithms*, 74(3):905–924.

Elfving, T., Nikazad, T., and Hansen, P. C. (2010). Semi-convergence and relaxation parameters for a class of sirt algorithms. *Electronic Transactions on Numerical Analysis*, 37:321–336.

Fernández, J.-J., Gordon, D., and Gordon, R. (2008). Efficient parallel implementation of iterative reconstruction algorithms for electron tomography. *Journal of Parallel and Distributed Computing*, 68(5):626–640.

Fessler, J. A., Ficaro, E. P., Clinthorne, N. H., and Lange, K. (1997). Grouped-coordinate ascent algorithms for penalized-likelihood transmission image reconstruction. *IEEE transactions on medical imaging*, 16(2):166–175.

Fessler, J. A. and Kim, D. (2011). Axial block coordinate descent (abcd) algorithm for x-ray ct image reconstruction. *Proceedings of Fully 3D Image Reconstruction in Radiology and Nuclear Medicine*, pages 262–265.

Flores, L. A., Vidal, V., Mayo, P., Rodenas, F., and Verdú, G. (2014). Parallel ct image reconstruction based on gpus. *Radiation Physics and Chemistry*, 95:247–250.

Forsythe, G. E. (1953). Solving linear algebraic equations can be interesting. *Bulletin of the American Mathematical Society*, 59(4):299–329.

Gao, Y., Biguri, A., and Blumensath, T. (2019). Block stochastic gradient descent for large-scale tomographic reconstruction in a parallel network. *arXiv preprint arXiv:1903.11874*.

Gao, Y. and Blumensath, T. (2017). A parallel ct reconstruction algorithm using partial row and column blocks of the system matrix. In *TOSCA*.

Gao, Y. and Blumensath, T. (2018a). Bsgd-tv: A parallel algorithm solving total variation constrained image reconstruction problems. *arXiv preprint arXiv:1812.01307*.

Gao, Y. and Blumensath, T. (2018b). A joint row and column action method for cone-beam computed tomography. *IEEE Transactions on Computational Imaging*, 4(4):599–608.

Gao, Y. and Blumensath, T. (2019). A parallel stochastic algorithm for large scale ct reconstruction. In *dXCT*.

Gholizadeh, S. (2016). A review of non-destructive testing methods of composite materials. *Procedia structural integrity*, 1:50–57.

Gilbert, P. (1972). Iterative methods for the three-dimensional reconstruction of an object from projections. *Journal of theoretical biology*, 36(1):105–117.

Gordon, G. and Tibshirani, R. (2012). Gradient descent revisited. *Optimization*, 10(725/36):725.

Gordon, R., Herman, G. T., and Johnson, S. A. (1975). Image reconstruction from projections. *Scientific American*, 233(4):56–71.

Grangeat, P. (1991). Mathematical framework of cone beam 3d reconstruction via the first derivative of the radon transform. In *Mathematical methods in tomography*, pages 66–97. Springer.

Gregor, J. and Benson, T. (2008). Computational analysis and improvement of sirt. *IEEE Transactions on Medical Imaging*, 27(7):918–924.

Gregor, J. and Fessler, J. A. (2015). Comparison of sirt and sqs for regularized weighted least squares image reconstruction. *IEEE transactions on computational imaging*, 1(1):44–55.

Guo, P. and Devaney, A. J. (2005). Comparison of reconstruction algorithms for optical diffraction tomography. *JOSA A*, 22(11):2338–2347.

Guo, W. and Chen, H. (2012). Improving sirt algorithm for computerized tomographic image reconstruction. In *Recent Advances in Computer Science and Information Engineering*, pages 523–528. Springer.

Ha, S. and Mueller, K. (2015). An algorithm to compute independent sets of voxels for parallelization of icd-based statistical iterative reconstruction. In *The 13th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*.

Hansen, P. C. (1992). Analysis of discrete ill-posed problems by means of the l-curve. *SIAM review*, 34(4):561–580.

Hansen, P. C. and O'Leary, D. P. (1993). The use of the l-curve in the regularization of discrete ill-posed problems. *SIAM journal on scientific computing*, 14(6):1487–1503.

He, N. and Buccafusca, L. (2016). Lecture 12: Coordinate descent algorithms. *Big Data Optimization*, page 7.

Herman, G. and Davidi, R. (2008). Image reconstruction from a small number of projections. *Inverse problems*, 24(4):1–17.

Herman, G. T. (2009). *Fundamentals of computerized tomography: image reconstruction from projections*. Springer Science and Business Media.

Herman, G. T., Lent, A., and Lutz, P. H. (1978). Relaxation methods for image reconstruction. *Communications of the ACM*, 21(2):152–158.

Hsieh, C. J., Chang, K. W., Lin, C. J., Keerthi, S. S., and Sundararajan, S. (2008). A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM.

Hsieh, J. (2003). Analytical models for multi-slice helical ct performance parameters. *Medical physics*, 30(2):169–178.

Hsieh, J., Nett, B., Yu, Z., Sauer, K., Thibault, J.-B., and Bouman, C. A. (2013). Recent advances in ct image reconstruction. *Current Radiology Reports*, 1(1):39–51.

Hu, H. (1999). Multi-slice helical ct: Scan and reconstruction. *Medical physics*, 26(1):5–18.

Huang, X., Liu, G., and Niu, Q. (2020). Remarks on kaczmarz algorithm for solving consistent and inconsistent system of linear equations. In *International Conference on Computational Science*, pages 225–236. Springer.

Hudson, H. M. and Larkin, R. S. (1994). Accelerated image reconstruction using ordered subsets of projection data. *IEEE transactions on medical imaging*, 13(4):601–609.

Islam, F. (2013). Insufficient ct data reconstruction based on directional total variation (dtv) regularized maximum likelihood expectation maximization (mlem) method. Master's thesis, Missouri University of Science and Technology, Missouri.

Jiang, M. and Wang, G. (2003). Convergence studies on iterative algorithms for image reconstruction. *IEEE Transactions on Medical Imaging*, 22(5):569–579.

Jin, X., Li, L., Chen, Z., Zhang, L., and Xing, Y. (2010). Anisotropic total variation for limited-angle ct reconstruction. In *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*, pages 2232–2238. IEEE.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323.

Kak, A. C. (1979). Computerized tomography with x-ray, emission, and ultrasound sources. *Proceedings of the IEEE*, 67(9):1245–1272.

Kamath, G., Ramanan, P., and Song, W.-Z. (2015). Distributed randomized kaczmarz and applications to seismic imaging in sensor network. In *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*, pages 169–178. IEEE.

Kamilov, U. S. (2016). Parallel proximal methods for total variation minimization. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4697–4701. IEEE.

Katsevich, A. (2002a). Analysis of an exact inversion algorithm for spiral cone-beam ct. *Physics in Medicine & Biology*, 47(15):2583.

Katsevich, A. (2002b). Theoretically exact filtered backprojection-type inversion algorithm for spiral ct. *SIAM Journal on Applied Mathematics*, 62(6):2012–2026.

Kazantsev, D., Lionheart, W., Withers, P., and Lee, P. (2013). Gpu accelerated 4d-ct reconstruction using higher order pde regularization in spatial and temporal domains. *Proc. CMSSE*, 3:843–852.

Kim, D. and Fessler, J. A. (2012). 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.

Kim, D., Ramani, S., and Fessler, J. A. (2013). Accelerating x-ray ct ordered subsets image reconstruction with nesterov's first-order methods. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 22–5.

Konečnỳ, J., Liu, J., Richtárik, P., and Takáč, M. (2015). Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255.

Konečnỳ, J., Qu, Z., and Richtárik, P. (2017). Semi-stochastic coordinate descent. *Optimization Methods and Software*, 32(5):993–1005.

Kongskov, R. and Dong, Y. (2017). Tomographic reconstruction methods for decomposing directional components. *arXiv preprint arXiv:1708.06912*.

Leventhal, D. and Lewis, A. S. (2010). Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pages 583–598.

Li, X., Ni, J., and Wang, G. (2005). Parallel iterative cone beam ct image reconstruction on a pc cluster. *Journal of X-ray science and technology*, 13(2):63–72.

Lionheart, B. (2013). What is sufficient data for stable ct reconstruction?

Liu, J., Wright, S. J., and Sridhar, S. (2014). An asynchronous parallel randomized kaczmarz algorithm. *arXiv preprint arXiv:1401.4780*.

Lu, Z. and Xiao, L. (2015). On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152(1-2):615–642.

Luo, Z.-Q. (1991). On the convergence of the lms algorithm with adaptive learning rate for linear feedforward networks. *Neural Computation*, 3(2):226–245.

Manzke, R., Koken, P., Hawkes, D., and Grass, M. (2005). Helical cardiac cone beam ct reconstruction with large area detectors: a simulation study. *Physics in Medicine & Biology*, 50(7):1547.

Moorman, J. D., Tu, T. K., Molitor, D., and Needell, D. (2020). Randomized kaczmarz with averaging. *arXiv preprint arXiv:2002.04126*.

Natterer, F. (2001). *The mathematics of computerized tomography*. SIAM.

Needell, D. (2010). Randomized kaczmarz solver for noisy linear systems. *BIT Numerical Mathematics*, 50(2):395–403.

Needell, D. and Tropp, J. A. (2014). Paved with good intentions: Analysis of a randomized block kaczmarz method. *Linear Algebra and its Applications*, 441:199–221.

Needell, D., Ward, R., and Srebro, N. (2014). Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025.

Needell, D., Zhao, R., and Zouzias, A. (2015). Randomized block kaczmarz method with projection for solving least squares. *Linear Algebra and its Applications*, 484:322–343.

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

Nitanda, A. (2014). Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, pages 1574–1582.

Oliveira, E. F., Melo, S. B., Dantas, C. C., Vasconcelos, D. A., and Cadiz, F. (2011). Comparison among tomographic reconstruction algorithms with a limited data. In *International Nuclear Atlantic Conference-INAC 2011*.

Othman, A. E., Afat, S., Brockmann, M. A., Nikoubashman, O., Brockmann, C., Nikolaou, K., and Wiesmann, M. (2016). Radiation dose reduction in perfusion ct imaging of the brain: A review of the literature. *Journal of Neuroradiology*, 43(1):1–5.

Palenstijn, W. J., Bédorf, J., Batenburg, K. J., King, M., Glick, S., and Mueller, K. (2015). A distributed sirt implementation for the astra toolbox. In *Proc. Fully Three-Dimensional Image Reconstruct. Radiol. Nucl. Med.*, pages 166–169.

Pan, X., Sidky, E. Y., and Vannier, M. (2009). Why do commercial ct scanners still employ traditional, filtered back-projection for image reconstruction? *Inverse problems*, 25(12):123009.

Parikh, N. and Boyd, S. (2014). Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102.

Plakhov, A. and Cruz, P. (2004). A stochastic approximation algorithm with step-size adaptation. *Journal of Mathematical Sciences*, 120(1):964–973.

Qin, Z., Scheinberg, K., and Goldfarb, D. (2013). Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, 5(2):143–169.

Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701.

Richtárik, P. and Takáč, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38.

Richtárik, P. and Takáč, M. (2016). Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484.

Rose, S., Andersen, M. S., Sidky, E. Y., and Pan, X. (2014). Application of incremental algorithms to ct image reconstruction for sparse-view, noisy data. In *3rd International Conference on Image Formation in X-Ray Computed Tomography*, pages 351–354.

Roux, N. L., Schmidt, M., and Bach, F. R. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in neural information processing systems*, pages 2663–2671.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rudin, L. I. and Osher, S. (1994). Total variation based image restoration with free local constraints. In *Proceedings of 1st International Conference on Image Processing*, volume 1, pages 31–35. IEEE.

Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268.

Sabne, A., Wang, X., Kisner, S. J., Bouman, C. A., Raghunathan, A., and Midkiff, S. P. (2017). Model-based iterative ct image reconstruction on gpus. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 207–220. ACM.

Sauer, K. and Bouman, C. (1993). A local update strategy for iterative reconstruction from projections. *IEEE Transactions on Signal Processing*, 41(2):534–548.

Sauer, K. D., Borman, S., and Bouman, C. A. (1995). Parallel computation of sequential pixel updates in statistical tomographic reconstruction. In *Image Processing, 1995. Proceedings., International Conference on*, volume 2, pages 93–96. IEEE.

Schindler, M., Batzle, M. L., and Prasad, M. (2017). Micro x-ray computed tomography imaging and ultrasonic velocity measurements in tetrahydrofuran-hydrate-bearing sediments. *Geophysical Prospecting*, 65(4):1025–1036.

Schmidt, M., Le Roux, N., and Bach, F. (2017). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112.

Schmidt, M., Roux, N. L., and Bach, F. R. (2011). Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in neural information processing systems*, pages 1458–1466.

Selesnick, I., Lanza, A., Morigi, S., and Sgallari, F. (2020). Non-convex total variation regularization for convex denoising of signals. *Journal of Mathematical Imaging and Vision*, pages 1–17.

Shalev Shwartz, S. and Tewari, A. (2011). Stochastic methods for l1-regularized loss minimization. *Journal of Machine Learning Research*, 12(Jun):1865–1892.

Sheng, X. and Chen, G. (2010). A note of computation for mp inverse a. *International Journal of Computer Mathematics*, 87(10):2235–2241.

Sidky, E., Duchin, Y., Pan, X., and Ullberg, C. (2011). A constrained, total-variation minimization algorithm for low-intensity x-ray ct. *Medical physics*, 38(7):S117–S125.

Sidky, E. Y., Kao, C., and Pan, X. (2006). Accurate image reconstruction from few-views and limited-angle data in divergent-beam ct. *Journal of X-ray Science and Technology*, 14(2):119–139.

Sidky, E. Y. and Pan, X. (2008). Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization. *Physics in medicine and biology*, 53(17):4777–4807.

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

Stiller, W. (2018). Basics of iterative reconstruction methods in computed tomography: a vendor-independent overview. *European journal of radiology*, 109:147–154.

Strohmer, T. and Vershynin, R. (2009). A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278.

Sznajder, R. (2016). Kaczmarz algorithm revisited. *Czasopismo Techniczne*.

Taina, I., Heck, R., and Elliot, T. (2008). Application of x-ray computed tomography to soil science: A literature review. *Canadian Journal of Soil Science*, 88(1):1–19.

Tanabe, K. (1971). Projection method for solving a singular system of linear equations and its applications. *Numerische Mathematik*, 17(3):203–214.

Tang, N. D., De Ruiter, N., Mohr, J., Butler, A. P., Butler, P. H., and Aamir, R. (2012). Using algebraic reconstruction in computed tomography. In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, pages 216–221. ACM.

Thibault, J.-B., Sauer, K. D., Bouman, C. A., and Hsieh, J. (2007). A three-dimensional statistical approach to improved image quality for multislice helical ct. *Medical physics*, 34(11):4526–4544.

Thompson, W. M. and Lionheart, W. R. (2014). Gpu accelerated structure-exploiting matched forward and back projection for algebraic iterative cone beam ct reconstruction. In *The Third International Conference on Image Formation in X-Ray Computed Tomography, 22-25 June 2014, Salt Lake City, Utah, USA.*

Tian, Z., Jia, X., Yuan, K., Pan, T., and Jiang, S. B. (2011). Low-dose ct reconstruction via edge-preserving total variation regularization. *Physics in Medicine & Biology*, 56(18):5949.

Tikhonov, A. N. (1963). On the solution of ill-posed problems and the method of regularization. In *Doklady Akademii Nauk*, volume 151, pages 501–504. Russian Academy of Sciences.

Turbell, H. (2001). *Cone-beam reconstruction using filtered backprojection.* PhD thesis, Linköping University Electronic Press.

van Aarle, W., Palenstijn, W. J., De Beenhouwer, J., Altantzis, T., Bals, S., Batenburg, K. J., and Sijbers, J. (2015). The astra toolbox: A platform for advanced algorithm development in electron tomography. *Ultramicroscopy*, 157:35–47.

Van Scoy, B., Freeman, R. A., and Lynch, K. M. (2017). The fastest known globally convergent first-order method for minimizing strongly convex functions. *IEEE Control Systems Letters*, 2(1):49–54.

Verhaeghe, J., Van De Ville, D., Khalidov, I., D'Asseler, Y., Lemahieu, I., and Unser, M. (2008). Dynamic pet reconstruction using wavelet regularization with adapted basis functions. *IEEE transactions on medical imaging*, 27(7):943–959.

Wang, G., Yu, H., and De Man, B. (2008). An outlook on x-ray ct research and development. *Medical physics*, 35(3):1051–1064.

Wang, H. and Banerjee, A. (2014). Randomized block coordinate descent for online and stochastic optimization. *arXiv preprint arXiv:1407.0107.*

Wang, J., Zeng, L., Wang, C., and Guo, Y. (2019). Admm-based deep reconstruction for limited-angle ct. *Physics in Medicine & Biology*, 64(11):115011.

Wang, J., Zhang, C., and Wang, Y. (2017a). A photoacoustic imaging reconstruction method based on directional total variation with adaptive directivity. *Biomedical engineering online*, 16(1):64.

Wang, S. and Liao, L. (2001). Decomposition method with a variable parameter for a class of monotone variational inequality problems. *Journal of optimization theory and applications*, 109(2):415–429.

Wang, X. (2008). Method of steepest descent and its applications. *IEEE Microwave and Wireless Components Letters*, 12:24–26.

Wang, X., Sabne, A., Kisner, S., Raghunathan, A., Bouman, C., and Midkiff, S. (2016). High performance model based image reconstruction. *ACM SIGPLAN Notices*, 51(8):1–12.

Wang, X., Sabne, A., Sakdhnagool, P., Kisner, S. J., Bouman, C. A., and Midkiff, S. P. (2017b). Massively parallel 3d image reconstruction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 3. ACM.

Xu, F., Xu, W., Jones, M., Keszthelyi, B., Sedat, J., Agard, D., and Mueller, K. (2010). On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on gpus. *Computer methods and programs in biomedicine*, 98(3):261–270.

Xu, Y. and Yin, W. (2015). Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716.

Ying, L., Xu, D., and Liang, Z.-P. (2004). On tikhonov regularization for image reconstruction in parallel mri. In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 1, pages 1056–1059. IEEE.

Yu, H. and Wang, G. (2009). Compressed sensing based interior tomography. *Physics in medicine and biology*, 54(9):2791–2805.

Yu, Z., Thibault, J.-B., Bouman, C. A., Sauer, K. D., and Hsieh, J. (2007). Nonhomogeneous updates for the iterative coordinate descent algorithm. In *Electronic Imaging 2007*, pages 64981B–64981B. International Society for Optics and Photonics.

Yu, Z., Thibault, J.-B., Bouman, C. A., Sauer, K. D., and Hsieh, J. (2010). Fast model-based x-ray ct reconstruction using spatially nonhomogeneous icd optimization. *IEEE Transactions on image processing*, 20(1):161–175.

Yu, Z., Thibault, J.-B., Sauer, K., Bouman, C., and Hsieh, J. (2006). Accelerated line search for coordinate descent optimization. In *2006 IEEE Nuclear Science Symposium Conference Record*, volume 5, pages 2841–2844. IEEE.

Zaccone, G. (2015). *Python parallel programming cookbook*. Packt Publishing Ltd.

Zhang, A. and Gu, Q. (2016). Accelerated stochastic block coordinate descent with optimal sampling. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2035–2044. ACM.

Zhang, S., Choromanska, A. E., and LeCun, Y. (2015). Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693.

Zhang, Y., Chan, H.-P., Sahiner, B., Wei, J., Goodsitt, M. M., Hadjiiski, L. M., Ge, J., and Zhou, C. (2006). A comparative study of limited-angle cone-beam reconstruction methods for breast tomosynthesis. *Medical physics*, 33(10):3781–3795.

Zhao, S.-Y. and Li, W.-J. (2016). Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI*, pages 2379–2385.

Zhao, T., Yu, M., Wang, Y., Arora, R., and Liu, H. (2014). Accelerated mini-batch randomized block coordinate descent method. In *Advances in neural information processing systems*, pages 3329–3337.

Zheng, J., Saquib, S. S., Sauer, K., and Bouman, C. A. (2000). Parallelizable bayesian tomography algorithms with rapid, guaranteed convergence. *IEEE Transactions on Image Processing*, 9(10):1745–1759.

Zou, Y. and Pan, X. (2004). Image reconstruction on pi-lines by use of filtered backprojection in helical cone-beam ct. *Physics in Medicine & Biology*, 49(12):2717.

Zouzias, A. and Freris, N. M. (2013). Randomized extended kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793.