IMAGE RECONSTRUCTIONS OF COMPRESSED SENSING MRI WITH MULTICHANNEL DATA

A Dissertation

by

CHING-HUA CHANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Jim Ji
Committee Members,	Raffaella Righetti
	Henry Pfister
	Mary P. McDougall
Head of Department,	Miroslav M. Begovic

May 2016

Major Subject: Electrical Engineering

ABSTRACT

Magnetic resonance imaging (MRI) provides high spatial resolution, high-quality of soft-tissue contrast, and multi-dimensional images. However, the speed of data acquisition limits potential applications. Compressed sensing (CS) theory allowing data being sampled at sub-Nyquist rate provides a possibility to accelerate the MRI scan time. Since most MRI scanners are currently equipped with multi-channel receiver systems, integrating CS with multi-channel systems can further shorten the scan time and also provide a better image quality. In this dissertation, we develop several techniques for integrating CS with parallel MRI.

First, we propose a method which extends the reweighted l_1 minimization to the CS-MRI with multi-channel data. The individual channel images are recovered according to the reweighted l_1 minimization algorithm. Then, the final image is combined by the sum-of-squares method. Computer simulations show that the new method can improve the reconstruction quality at a slightly increased computation cost.

Second, we propose a reconstruction approach using the ubiquitously available multi-core CPU to accelerate CS reconstructions of multiple channel data. CS reconstructions for phase array system using iterative l_1 minimization are significantly time-consuming, where the computation complexity scales with the number of channels. The experimental results show that the reconstruction efficiency benefits significantly from parallelizing the CS reconstructions, and pipelining multi-channel data on multi-core

processors. In our experiments, an additional speedup factor of 1.6 to 2.0 was achieved using the proposed method on a quad-core CPU.

Finally, we present an efficient reconstruction method for high-dimensional CS MRI with a GPU platform to shorten the time of iterative computations. Data managements as well as the iterative algorithm are properly designed to meet the way of SIMD (single instruction/multiple data) parallelizations. For three-dimension multi-channel data, all slices along frequency encoding direction and multiple channels are highly parallelized and simultaneously processed within GPU. Generally, the runtime on GPU only requires 2.3 seconds for reconstructing a simulated 4-channel data with a volume size of 256×256×32. Comparing to 67 seconds using CPU, it achieves 28 faster with the proposed method. The rapid reconstruction algorithms demonstrated in this work are expected to help bring high dimensional, multichannel parallel CS MRI closer to clinical applications.

Dedicated to my father, Sheng-Ping Chang (1924-2013), who suffered from WWII in the battle of Asia and retreated from Mainland China to Taiwan. He believed that only education can help people get out of poverty and have a better life. I would not finish this work without his support, encouragement, and endless love.

ACKNOWLEDGEMENTS

It is a long journey for me to earn the doctorate and to accomplish the dissertation. Undoubtedly, this arduous goal cannot be achieved without many people's help. First and the foremost, I would like to express my heartfelt gratitude to my family for their constant support and encouragement. The deepest thanks go to my parents, who are always proud of me and never lose faith in me whenever I have difficulties. Their trust gives me a lot of confidence and courage to take challenges. I am truly grateful to them. Thanks also go to my brothers and sisters-in-law that they are supportive of me and the family, especially in my absence from our father's last few years. In their unfailing care, I still could spend time and energy pursuing goals. Most important of all, it's the family who convinced me that the scene will be different after passing through a valley as I lost my persistence. I wish that I could picture the open scenery of this achievement and share with them since they accompany me in spirit along the way.

Certainly, I am sincerely thankful to my committee chair, Dr. Ji for giving me opportunities to do MRI research and also for his support, guidance and much patience as I sometimes got stranded. Thanks also go to my committee members, Dr. Righetti, and Dr. McDougall, who ever gave me valuable comments and suggestions. I also would like to thank Dr. Pfister, who is my husband's committee chair and my committee member, for his support, advice, and for showing many kindnesses to my family.

I also would like to thank my labmates, Ying, and Shuo, who are brilliant, outstanding and always willing to help. I benefited a lot in class and in the lab from our

discussions and their suggestions. It's definitely my pleasure to work with them. Thanks also go to another labmate, Aydin, who is kind and responsible. Although we didn't get many chances to spend time together, he offered a great help in my absence from the campus. I am grateful to him.

Last but not the least, I would like to express my utmost appreciation to my husband, Yung-Yih, who has been my classmate for 24 hours a day and 7 days a week. I always gain much from our discussions. Besides, he sometimes plays roles of both father and mother to our two daughters; sometimes, he plays roles of both partner and tutor to me. Although I occasionally feel "appropriate pressure" from him, it has been the spur for me to pursue the goal. Thanks also go to my two daughters, Alexis and Alina, who have been the sweetest and greatest cheerleader. I wish to have 48 hours a day and spend a half of the time with our beloved daughters.

I deeply appreciate that these people accompany, help, and keep me going. Because of them, I have seen and cherished different scenes in this journey, and finally, this can come to fulfillment in the end.

NOMENCLATURE

MRI	Magnetic Resonance Imaging
NMR	Nuclear Magnetic Resonance
CS	Compressed Sensing
FOV	Field of View
RF	Radio Frequency
SS	Slice Selection
FE	Frequency Encoding
PE	Phase Encoding
PI	Parallel Imaging
SENSE	Sensitivity Encoding
TV	Total Variation
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FD	Finite Difference
CPU	Central Processing Unit
GPU	Graphics Processing Unit
SMX	Streaming Multiprocessors
CG	Conjugate Gradient
ADMM	Alternating Direction Method of Multipliers
SNR	Signal-to-Noise Ratio

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION OF MRI

Magnetic Resonance Imaging

Magnetic resonance imaging (MRI), which is a non-invasive technique, has steadily developed in clinic applications and played an important role in assessing brain disease, cardiac problem, spinal disorder, and angiography. Unlike Computer Tomography (CT), Positron Emission Tomography (PET), or X-ray, MRI is a non-ionizing modality, giving high dimension capabilities, high spatial resolution and excellent contrast of soft tissue. In the early 1970s, MRI was developed from the technique of nuclear magnetic resonance (NMR), which has been used in analyzing chemical compositions for many years. Basically, in a strong magnet, nucleuses are magnetically polarized. By applying a weak radio frequency (RF), which is a nonionizing electromagnetic radiation, causing the protons precess coherently, the NMR signals arise from coils detectors where the voltage is induced by the sum of all the precessing protons. Because the only RF is applied to the body and the word “nuclear” is not relative to radioactivity, there are no damage or alterations to cellular DNA in MRI scanning process. Therefore, it is widely used and recommended when compared to CT [1-3].

From the fundamental of NMR, a quantum physical phenomenon, nuclei with an odd number of protons, neutrons or both has nuclear magnetism. That is because it has an electrical charge, spins very fast, and thus produces a noticeable magnetic field. Approximately, 2/3 of all stable nuclei can be regarded as little magnets. Since water

composes the biggest part of human bodies, most clinical MRI scanning use abundant hydrogen atoms (^{1}H) to produce NMR signals. The protons of hydrogen with +1 electrical charge exhibit the property of the magnetic dipole moment, causing proton spinning. Commonly, the spinning protons can be regarded as little magnets and referred to as spins. In normal condition, the spins point in random directions. By applying a strong static magnetic field, B_0, whose direction is conventionally defined as the z-axis, spinning protons align with or against the field. The excess number of spinning protons, which align with the field is proportional to B_0, usually ranging from 1.5 to 3 Teslas. The stronger the static magnetic field is placed, the higher signal to noise ratio is gathered from MRI scanner. That is because more excess protons contribute to the MRI signal. The spinning protons precess about the z-axis of the external B_0. The frequency of precession is proportional to the strength of B_0. This is known as the Larmor equation,

$$\vec{B}(\vec{r}) = (B_0 + \vec{G} \cdot \vec{r})\hat{a}_z \qquad (1.1)$$

where ω_0 is so-called Larmor frequency or resonance frequency, and γ is the constant gyromagnetic ratio to every ^{1}H atom, equal to 267.52×10^6.

Another field, commonly called, B_1, is an electromagnetic radio frequency (RF) pulse. The short-lived time-varying RF pulses, whose orientation is perpendicular to B_0, are applied at the resonance frequency and absorbed by the spinning protons. It causes spins tip down toward x-y plane. Thus, the orientation, strength, and duration of RF pulses are precisely designed to control the tip angle, α. After the spins spiral down and RF pulses are completely transmitted, the rotating magnets cause the varying magnetic field and produce electromagnetic radiation. The excited spins tend to return to the original

equilibrium, releasing the absorbed RF energy. They realign instantly toward the orientation of B_0 with the rate of T_1 and T_2 relaxation, which will be discussed later in Block equation. Thus, the retransmitted RF energy from the excited nuclei can be received and measured by the coils in MRI scanner.

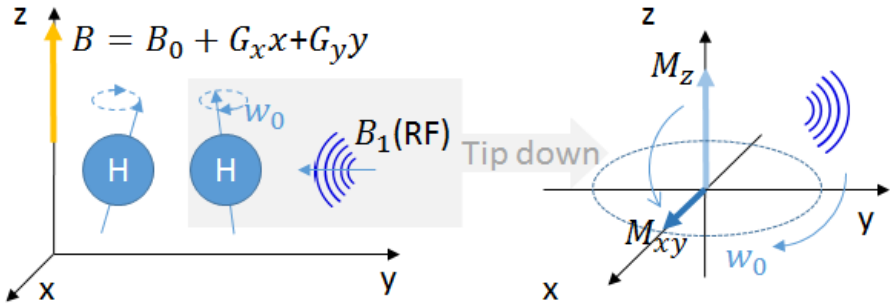


Figure 1.1 Illustration of protons precession under the static B_0 field and the effect when applying electromagnetic RF pulses.

The last important elements added in MRI are the gradients of magnetic field $G = [G_x, G_y, G_z]^T$, whose orientation is the same with B_0, causing the strength of magnetic field changing along x, y or z directions. This is accomplished by the gradient system, consisting three orthogonal gradient coils and producing a time-varying magnetic field. The gradient is zero at z=0, has odd symmetry about z-axis and is usually constant throughout the field of view (FOV), creating a linear variation in the magnetic field and the Larmor frequency with different positions. Therefore, the total static field is defined by

$$\vec{B}(\vec{r}) = (B_0 + \vec{G} \cdot \vec{r})\hat{a}_z$$

Where $\vec{r} = [x, y, z]^T$ represents the spatial position. Thus, eq.(1. 1) becomes as follows.

$$w(\vec{r}) = \gamma(B_0 + \vec{G} \cdot \vec{r})$$

This means that Larmor frequency varies according to the different spatial position. One can scan a slice of an object by selecting the orientation of RF pulse and tuning the RF signal to appropriate Larmor frequency. For example, one would like to select the imaging plane at $z = z_1$, then the frequency of RF pulse is chosen equal to $w_1 = w_0 + \gamma G_z z_1$ as shown in Figure 1.2. This leads to the spins in the plane being tipped down and leaving those spins off the plane unaffected.

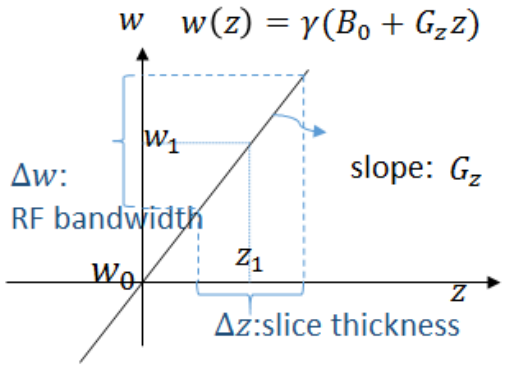


Figure 1.2 Illustration of slice selection relative to Larmor equation.

Block equation is an important formula describing the behavior of spins in the magnetic field. In classical mechanics, each spin has its own magnetic dipole moment, randomly distributed around the precession cone, and the abundant spins in bulk material contribute the magnetization vector, which is the sum of individual dipole moments.

Considering the effect of a magnetic field on these spins, the motion of the magnetization vector obeys the Block equation as follows,

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B} = \gamma\vec{M} \times (\vec{B}_0 + \vec{B}_1)$$

where $\vec{B}$ is the total magnetic field including $\vec{B}_0$ and $\vec{B}_1$ because magnetization vector is affected by any magnetic field. When applying RF pulses and creating $\vec{B}_1$ field, the magnetization vector is tipped away from the equilibrium state as shown in Figure 1.1. The magnetization defined as

$$\vec{M}(t) = [\vec{M}_z(t), \vec{M}_x(t), \vec{M}_y(t)]^T$$

$$= [M_z^0, M_x^0 cos\gamma B_0 t, -M_x^0 sin\gamma B_0 t]^T$$

precesses around z-axis in a left-hand sense at the equilibrium state, where $M_z^0 = \vec{M}_z(t=0) = M_0$, $M_x^0 = \vec{M}_x(t=0) = 0$,and $M_y^0 = \vec{M}_y(t=0) = 0$. It is easier to observe the motion of magnetization vector in rotating frame in absence of other field perturbations, such as $\vec{B}_1$ or magnetic field inhomogeneities. The new axes, x' and y', in the rotating frame rotate about the z-axis with a rotational frequency $\vec{w}_{rf} = \vec{w}_0 = -\gamma\vec{B}_0$, so that there is no precession about z-axis. The Block equation in rotating frame becomes as follows,

$$\frac{\partial\vec{M}_{rf}}{\partial t} = \gamma\vec{M}_{rf} \times \vec{B}_{eff}$$

where $\vec{B}_{eff} = \vec{B}_{rf} + \frac{\vec{w}_{rf}}{\gamma} + \vec{B}_1'$, $\vec{B}_{rf} = \vec{B}_0$, and the first and second term will be canceled in ideal z directed magnetic field. The left term, $\vec{B}_1'$, is the applied RF field in the rotating frame. Therefore, the magnetization vector will precess around $\vec{B}_1'$ in a left-handed sense,

causing the longitudinal magnetization $\vec{M}_z$ decrease and the transverse magnetization $\vec{M}_{xy}$ increase as shown in Figure 1.1.

During the magnetization vector return to the equilibrium state, it undergoes T_1 and T_2 relaxation processes. The $\vec{M}_z$ component starts to recoever back to M_0 and is governed by the spin-lattice relaxation time (T_1), due to protons losing the energy and heating up the surrounding tissue. Therefore, the rate of heating must be considered when designing RF excitation pulses. The magnitude of $\vec{M}_z(t)$ after the spins are excited by the RF pules is given by

$$\vec{M}_z(t) = M_0(1 - (1 - cos\alpha) \cdot e^{-t/T_1})$$

Similar to T_1 relaxation, T_2 decay is relative to magnetization in the transverse plane. The spins are all in phase when they are tilted down. As soon as the RF pulse is turned off, they lose phase coherence because each spin precesses with difference frequencies. The transverse magnetization decays due to spin-spin relaxation time (T_2) is describe by an exponential curve as follows

$$\vec{M}_{xy}(t) = M_0 \cdot e^{-t/T_2}$$

Therefore, considering the relaxation processes, the Block equation should be modified by the following,

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B} - \frac{M_x\hat{a}_x + M_y\hat{a}_y}{T_2} - \frac{(M_z - M_0)\hat{a}_z}{T_1}$$

where T_1 and T_2 are unique to different tissues, so that they can be used to differentiate between different types of tissues in clinical imaging.

We have partially introduced about gradient encoding for slice selection (SS), which is often along the z direction. The spatial variation in the magnetic field along the other two directions (x, and y) can distinguish protons at different locations. Once the slice is chosen, how to encode gradients along x-y plane for spatial information is relative to the sampling of encoded data in frequency domain. The most common encoding method is 2D Fourier, i.e. Cartesian sampling. Since NMR signal is received based on Faraday's law, and we have a time-varying magnetic field, $\vec{M}_{xy}(t)$, we will get the induced voltage, $v(t) = j\sqrt{2}w\Delta V M_{xy}^{0} B_{1t} e^{-\frac{t}{T_2}} e^{jwt}$, where B_{1t} denotes the effective coils sensitivity, ΔV is the voxel size containing magnetization. After proper demodulation and digitized, the signal, $S(t)$, received from all excited spins is obtained by integrating in x, y, and z,

$$S(t) = \iint \frac{1}{\sqrt{2}} j w_0 \Delta z M_{xy}^{0} B_{1t} e^{-\frac{t}{T_2}} e^{j(w - w_{mix})t} dxdy \tag{1. 2}$$

where w_{mix} is the frequency for quadrature demodulator, and is usually set to Larmor frequency, $w_{mix} = w_0$. Assume that we have ideal constant gradients. Applying gradients along x and y, known as frequency encoding (FE) and phase encoding (PE) respectively shown in Figure 1.3, a general form of the equation becomes as follows.

$$S(t) = \iint \frac{1}{\sqrt{2}} j w_0 \Delta z M_{xy}^{0} B_{1t} e^{-\frac{t}{T_2}} e^{j((w_0 + \gamma G_x x) - w_{mix})t + j\gamma G_y y T_p} dxdy \tag{1. 3}$$

By substituting$k_x = \gamma G_x t$, and $k_y = \gamma G_y T_p$, $S(t)$ turns out only relative to k_x and k_y, which are commonly called k-space in MRI. The equation is simplified and becomes as follows.

$$S(k_x, k_y) = \iint I(x,y) e^{jk_x x} e^{jk_y y} dxdy \qquad (1.4)$$

Therefore, the image $I(x, y)$ can be reconstructed by applying the 2D discrete inverse Fourier transform.

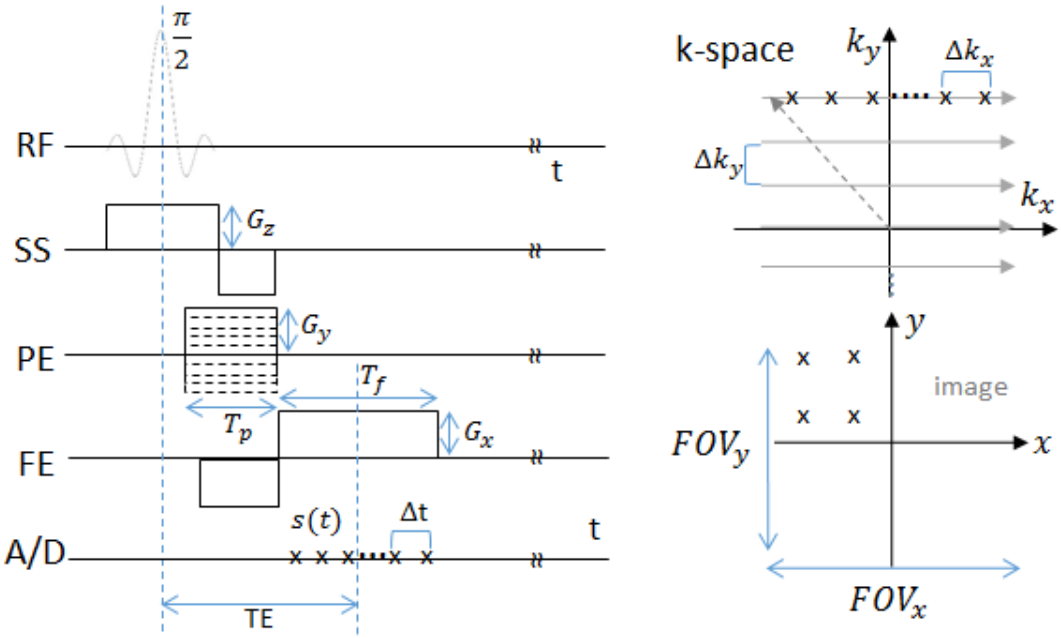


Figure 1.3 Illustration of gradient echo sequence for Cartesian sampling. The pulse diagram is repeated except that the amplitude of G_y is changed one step smaller. The signal $s(t)$ is digitized and shown the trajectory in k-space. The image is finally recovered by taking 2D inverse Fourier transform.

Figure 1.3 shows the pulse diagram of gradient echo sequence in MRI scanning. First, a sinc-like RF pulse of finite duration is applied to select one slice of the desired image. The bandwidth of the RF pulse controls the slice thickness of the desired image, exciting the protons within the slice. The magnetization vector has been tipped down 90 degrees by the pulse. Because the absence of slice selection gradient, which is usually referred to as logical z direction, the spins dephase with different frequencies, which can be reversed by applying a gradient with opposite sign. The excitation is considered to act on the spins at the midpoint of the RF pulse, and then dephasing starts instantaneously. The amount of dephasing is the product of gradient and the duration. Therefore, making the area equal can cancel the dephasing effect and all the spins are in phase by the end of slice select (SS) gradient. Second, the gradients of phase encoding (PE) and frequency encoding (FE), also known as readout channel, follow after the spins are excited. The first phase encoding gradient for the first acquired view is commonly the largest. The positive largest PE gradient and the negative FE gradient makes the sampling path go toward the up-left corner of k-space. Note that because the spins start to dephase at the end of SS gradient, it is better to avoid excessive long echo time (TE) shown in Figure 1.3. In order to shorten TE, practically, SS rephasing, PE gradient, and readout gradient dephasing can occur simultaneously. It means that the gradients that affect the phase of spins are put together. Thus, the data is acquired right after SS rephasing. The NMR signal is digitized by the analog-to-digital converter, and is acquired during the absence of positive FE gradient in that sampling begins from left to right in the k-space. After acquiring a line of samples in the k-space, the PE gradient is one step smaller so that the sampling path goes

to next line. The FE gradient repeats until the PE gradient gets to the amplitude with a negative sign. The acquisition is also repeated for different amplitudes of PE. Figure 1.3 demonstrates the conventional Cartesian sampling in the k-space, and the desired image can be reconstructed by taking 2D inverse Fourier transform.

Based on the Nyquist sampling theory, the sampling rate is relative to the resolution of acquired data in k-space, which denotes as Δk_x and Δk_y. From Figure 1.3 and eq.(1. 3), one knows that $\Delta k_x = \gamma G_x \Delta t$ and $\Delta k_y = \gamma \Delta G_y T_p$ for constant gradients. Therefore, the k-space resolution along frequency encode is controlled by the amplitude of FE gradient and the sampling rate, and the resolution along phase encode direction is controlled by the duration and the increment of PE gradient. The maximum frequency that we can sample is limited according to the equations, $\left|k_{x,max}\right| \leq \frac{\gamma G_x T_f}{2}$, and $\left|k_{y,max}\right| \leq \gamma G_y T_p$ based on the Nyquist sampling theory. To determine an appropriate range of MRI images (FOV) and the resolution of the acquired data without spatial aliasing, the gradient strength and the parameters for pulse sequence should be carefully chosen. According to discrete Fourier transform, field of view (along x and y) are determined by $FOV_x = \frac{2\pi}{\Delta k_x}$ and $FOV_y = \frac{2\pi}{\Delta k_y}$, and is inversely relative to the sampling rate and the step of PE gradient. All these parameter settings should be bounded by the limited bandwidth, i.e. $2\left|k_{x,max}\right|$ and $2\left|k_{y,max}\right|$. Because field of view is also defined as $FOV_x = N\Delta x$ and $FOV_y = N\Delta y$, the resolution of reconstructed image is determined by $\Delta x = \frac{2\pi}{N\gamma G_x \Delta t}$ and $\Delta y = \frac{2\pi}{N\gamma \Delta G_y T_p}$. These parameters and the relation are illustrated in Figure 1.3. For different sampling

methods, such as radial sampling and spiral sampling, the gradient forms vary differently. Therefore, a more general form of equation (is shown as follows,

$$S(t) = \int_{\vec{r}} I(\vec{r})\, e^{jk(t)\cdot\vec{r}} d\vec{r} \qquad (1.5)$$

where $\vec{r} = [x, y, z]^T$, $\vec{k}(t) = \gamma \int_0^t G(\vec{r}, \tau) d\tau$, and $\vec{B}(\vec{r}, t) = (B_0 + \int_0^t \vec{G}(\tau) \cdot \vec{r}\, d\tau)\hat{a}_z$. $\vec{k}(t)$ is known as the k-space trajectory. Essentially, the position in k-space can be located by controlling the duration of gradients. By precisely designing $G(t)$ and sample times, we can sample 2D or 3D, and uniform or non-uniform k-space data.

Usually, the temporal resolution of MRI imaging is much lower than CT or ultrasound. Because the magnitude of gradients is usually bounded for safety reason and limited by the physical hardware, we cannot increase the number of steps and the magnitude of PE gradient arbitrarily. On the other hand, it is more flexible to change the sampling rate of the analog to digital converter. Therefore, it is easier to achieve large FOV along the frequency encode direction without affecting the acquisition time, and the frequency encoding direction is typically chosen as the longest dimension when scanning MRI image in order to avoid aliasing. However, larger FOV or smaller size of voxels along phase encode direction requires more scan lines, which implies the total scan time will also increase. In Cartesian or radial trajectories, one RF excitation can usually generate one line of data with constant gradients. In general, it takes 2 to 10 minutes to generate MRI images. This limits the clinical applications. For instance, children or patients who have acute pain or claustrophobia may not be able to hold still. Besides, the signal-to-noise ratio (SNR) of MRI images is also proportional acquisition time. It is

difficult to distinguish tissue contrast from the background noise when SNR is too low. Therefore, to shorten the scan time, to reduce the motion blur and to increase patient's comfort has been an important goal of technical development.

Phase Array Receiver System

Since the speed of data acquisition has continued an important issue in MRI, many efforts were spent on the techniques of RF pulse sequence design and fast gradient switching. Another breakthrough in MRI is the development of multiple RF coils, which can receive data in parallel.

Phased array is one type of MRI RF receiver system, which has multi-channel coils and generally operate as "receive-only" device. Since each channel has its own receiver coil, it obviously provides the advantage of speeding up scan time, increase patient comfort and reduce image artifacts. Besides, array coils can gather more signals, and give more information. The term, phased array in MRI, was inspired from antenna theory. Many small antennas are grouped together to reduce noises and enhance overall signals. Another advantage is to expand the anatomical coverage and to achieve a large field of view, for instance, spine imaging or three-dimension angiogram. Therefore, with phased array receiver system, MRI scan has been suggested rather than the high radiation CT in some clinical applications.

The design and its application of coil arrays in MRI have been developing rapidly for past 35 years. Dating back to 1980's, switchable coil arrays were first used. Three single-segment spine coils were coupled together, covering the patient and extending FOV

[4-6]. Because receiver channels were much more complicated and expensive, early MRI scanner was usually equipped only one receiver channel and only one single segment spine coil could be chosen at each time. The switchable feature significantly shortens the scan time without moving the patient, even now this feature still remain in the modern design of surface coils. That is because coil elements are much cheaper than the receiver channels, which consist of amplifiers digitizing circuit design and computation module to reconstruct the acquired signals. Currently, the number of receiver channels is still much less than the number of coil elements.

In next advanced configuration of coil arrays, several small coils are combined together, which is known as phased array[7, 8]. The data are acquired simultaneously and fed independently into different receiver channels. Each coil has its own sensitive reception volume according to a variable sensitivity profile, which depends on the distance from the coil receiver. Originally, the MR phased array employed coils with, usually, overlapping sensitivity profile. As the design of array coils evolved, some parallel MRI reconstruction techniques, known as parallel imaging, has been developed rapidly in the last decade. In parallel imaging, it generally avoids the sensitivity profiles overlapping between coils because coil elements should not have magnetic interference and distinct sensitivity profiles of coils are preferred [9]. Therefore, not all phased array receivers are suitable for parallel MRI techniques although all coil elements used in parallel imaging are phased arrays. Parallel imaging techniques take advantage of local coils reception pattern and exploit the additional information of spatial encoding to further improve the spatial and temporal resolutions, and to increase SNR.

Ideally, assume that the noises from all channel have equal variances, ignoring correlations across channels, P independent phased array coils can improve SNR by a factor of $\sqrt{P}$. In reality, coil arrays are coupled and the induced magnetizations actually make noises correlated. Although it is hard to achieve such a large gain, a number of group explored the applications of coil arrays for possibility of fast imaging and better image quality[10, 11].

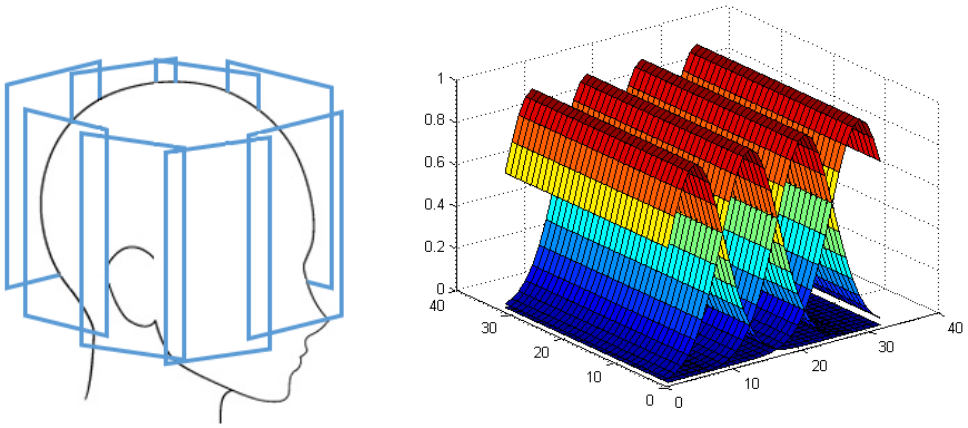


Figure 1.4 (a) Illustration of eight-channel phased array. (b) Four-channel sensitivity profiles – shifted linear Gaussian sensitivity.

Figure 1.4(a) shows eight-channel head array coils. According to eq.(1. 5), assume that the phased array with a number of L coils receives all signals simultaneously. When all relaxations are not considered, the signal received from the l^{th} coil can be formulated as follows.

$$S_l\left(G_y{}^i, G_z{}^i, t\right) = \int_{\vec{r}} I(\vec{r}) W_l(\vec{r}) P^L(\vec{r}) e^{j\gamma(G_x{}^i x t + G_y{}^i y \tau_1 + G_z{}^i z \tau_2)} d\vec{r} \quad (1.6)$$

Here, $\vec{r} = [x, y, z]^T$ and $W_l(\vec{r})$ is the function of complex 3-dimension sensitivity profile for the l^{th}coil, where $l = 1, \dots, L$. $P^L(\vec{r})$ represents the RF selective profile during the l^{th} excitation. Frequency encoding is along x direction, and τ_1 and τ_2 respectively denote the duration of phase encoding gradients along y and z directions. The index i corresponds to the i-th acquisition, so $G_x{}^i$, $G_y{}^i$ and $G_z{}^i$ represent the applied gradients during each data acquisition. With Fourier sampling, the parameter $P^L(\vec{r})$ remains constant for all value of l, and $G_x{}^i$ is usually constant for each data acquisition. We take 2-dimension case for specific instance. Note that the phased array coils have the effect of non-uniform spatial encoding as shown in Figure 1.4(b), which illustrates linear Gaussian functions for the 2-dimension sensitivity profile. In 2D Cartesian sampling, a slice in the 3D volume is selected by the RF excitation. That means the parameter $P^L(\vec{r})$ is set to 1, and 0 outside the desired slice. Assume that the preferred slice is at z_0, then $P^L(\vec{r})$ is equal to 1 as $z = z_0$. The general eq.(1. 6) could be simplified for the scanned slice at z_0,

$$S_l\left(G_y{}^i, t\right) = \iint I(x, y) W_l(x, y) e^{j\gamma(G_x x t + G_y{}^i y \tau)} dx dy \quad (1.7)$$

where G_x is constant for every acquisition because of 2D Cartesian sampling. After digitizing the continuous Fourier equation, the index of x and y in spatial domain are

replaced by the index of m and n. With changing the variables, let $k_x = \gamma G_x$, which is a constant, and ${k_y}^i = \gamma {G_y}^i \tau$, which is a variable. Then eq. (1. 7) become

$$S_l\left({k_y}^i, t\right) = \sum_{m=1}^{M} \sum_{n=1}^{N} I(m,n) W_l(m,n) e^{j(k_x m t + {k_y}^i n)} \quad (1.\ 8)$$

Here, t, m and n are discrete variables. The maximum value of sampled points for t is equal to the number of sample points along frequency encoding direction, M. If there are N steps along phase encoding direction, then eq.(1. 8) can be formulated as a matrix form,

$$\begin{bmatrix} S_l(G_y{}^1, 1) \\ S_l(G_y{}^1, 2) \\ \vdots \\ S_l(G_y{}^1, M) \\ \vdots \\ S_l(G_y{}^N, 1) \\ \vdots \\ S_l(G_y{}^N, M) \end{bmatrix}$$

$$= \begin{bmatrix} W_l(1,1)e^{j(k_x 1 + k_y{}^1 1)} & \cdots & W_l(M,N)e^{j(k_x M + k_y{}^1 1)} \\ W_l(1,1)e^{j(k_x 1 \cdot 2 + k_y{}^1 1)} & \cdots & W_l(N,N)e^{j(k_x M \cdot 2 + k_y{}^1 1)} \\ \vdots & \cdots & \vdots \\ W_l(1,1)e^{j(k_x 1 \cdot M + k_y{}^1 1)} & \cdots & W_l(M,N)e^{j(k_x M \cdot M + k_y{}^1 1)} \\ \vdots & \cdots & \vdots \\ W_l(1,1)e^{j(k_x 1 + k_y{}^N N)} & \cdots & W_l(M,N)e^{j(k_x M + k_y{}^N N)} \\ \vdots & \cdots & \vdots \\ W_l(1,1)e^{j(k_x 1 \cdot M + k_y{}^N N)} & \cdots & W_l(M,N)e^{j(k_x M \cdot M + k_y{}^N N)} \end{bmatrix} \quad (1.9)$$

$$\cdot \begin{bmatrix} I(1,1) \\ I(2,1) \\ \vdots \\ I(M,1) \\ \vdots \\ I(1,N) \\ \vdots \\ I(M,N) \end{bmatrix}$$

The vector on the left-hand side is the time-varying signal received from the l-th channel, which is arranged according to phase encoding step from 1 to N. The size of the acquisition system is $M \times N$ by $M \times N$. If there are L coils, the system becomes L by $M \times N$ by $M \times N$. Therefore, it is computational inefficient to reconstruct image I by inverting

the equations directly. However, sensitivity profile is regarded as an encoding matrix in spatial domain. By applying the efficient 2D Fourier transform to the acquired matrix, we can obtain $[W_1 I \quad W_2 I \quad \ldots \quad W_L I]$ from all channel data. The final image can be reconstructed in the least square sense, also known as root sum-of-square method as follows.

$$\hat{I} = ({W_1}^2 + {W_1}^2 + \cdots + {W_L}^2) I \tag{1. 10}$$

This solution is the optimal combination when the coil sensitivities are unknown, as proven in [7, 12]. The composite reconstructed image has high SNR, which is asymptotically as same as the result of maximum ratio combining based on the sensitivity profiles are perfectly known. Therefore, among advanced image reconstruction methods for phased array, such as parallel imaging usually, root sum-of-square method is usually used as a benchmark for comparisons.

Currently, one way to collect coil sensitivities is to gather from separate scans. It assumes that coil sensitivities are time invariant as the receiver coils are fixed. A calibration scan is processed prior to the desired image acquisition and the full k space data is obtained. The resulting images from all channels are weighted and reconstructed as $W_k(x, y) I(x, y)$. Using the body coil for the other scan, we can get $I(x, y)$because the sensitivity profiles of the body coil are assumed to be 1, $W(x, y) = 1$. Then, taking the ratio of two results from extra scans, we can get $W_k(x, y)$. Usually, the multiple coils are placed around the FOV and the combined sensitivities are designed to be homogeneous, so that adding all images are assumed to be $I(x, y)$, $I_s(x, y) = \sum_{l=1}^{L} I_l \approx I$. In this case, there is no need to take an extra scan by the body coil.

The second way is to estimate from low-resolution images, which is reconstructed from the central k-space data during the same scan[13]. Inaccurate estimations of coil sensitivities may cause artifacts if the coil sensitivities differ during the dynamic scans. Because sensitivity profiles vary smoothly according to spatial location, it is sufficient to obtain profiles from contiguous low-frequency data in k space. With the same assumption that the combined sensitivities are designed to be homogeneous, all channel-reconstructed images, $W_l I_{LF}$, from low frequency data are added together, resulting in a low-resolution image. $I_s(x,y) = \sum_{l=1}^{L} W_l I_{LF} \approx I_{LF}$. Similarly, taking the ratio of $W_l I_{LF}$ and I_s, we can estimate sensitivity profiles. With the phased array receiver coils and the estimation of sensitivity profiles, the further researches make it possible for extending FOV, speeding up acquisition time, reducing image artifacts and increasing SNR.

MRI Image Reconstructions

In order to shorten the acquisition time, the images are usually reconstructed from limited Fourier data instead of fully sampled data according to Nyquist sampling theory. This section provides a brief introduction to the inverse problem of the limited Fourier data from multi-channel. A wide variety of advanced image reconstruction techniques for multi-channel under-sampled data has emerged for past fifteen years. Among these advanced image reconstruction methods, parallel MR imaging, which uses an array of receiver coils to improve image quality and to increase SNR, is one of the most representative techniques. Limited Fourier data can be acquired more quickly, but it causes the spatial images to alias. Parallel imaging introduces algorithms for alias-free

reconstructions, one field of which regards the multiple receiver coils as a large linear system and solve the inverse problem of unfolding images, such as sensitivity encoding (SENSE) [14], SPACE RIP and PILS[13, 15, 16]. The other field is to interpolate missing lines and to synthesize a full k-space of the image from multi-channel data, such as SMASH [17] and GRAPPA [18]. Among these state-of-art methods, SENSE method, which reduces the aliasing of images by solving the equations of a linear system, can be combined with compressed sensing theory, and thus, there are many optimizations and extensions with SENSE method. Because SENSE and SPACE RIP provide an important way to formulate the system and regard the MRI reconstruction as an inverse problem, we briefly describe the two methods in this section.

SENSE solving a linear system relies on the information of estimated sensitivity profiles. For Cartesian SENSE, it requires equal-spaced down sampling along phase encoding lines according to the reduction factor R, which describes a ratio between numbers of fully sampled lines and the reduced lines. The reduced sampling in phase encoding direction causes a reduced FOV and alias artifacts in the spatial domain. Take R equal to 2 for example, the alias pixel in the spatial domain at (j, x) is actually a mixture of $I(j, x)$ and $I(j + N/2\,, x)$, which is located at another half of FOV. Thus, the spatially-aliased image at (j, x), resulting from taking the inverse Fourier transform of the down-sampled k space data, can be formulated as

$$a_1(j,x) = W_1(j,x)I(j,x) + W_1(j+N/2\,,x)I(j+N/2\,,x) \tag{1. 11}$$

The aliased image from the 1st coil is regarded as the sum of weighted pixels at a different position. Assume that the number of coils L is at least as larger as the reduction factor R, then the general form of all collected alias pixel at (j, x) from L channel coils can be expressed as,

$$\begin{bmatrix} a_1(j,x) \\ a_2(j,x) \\ \vdots \\ a_L(j,x) \end{bmatrix} = \begin{bmatrix} W_1(j,x) & \dots & W_1(j+N(R-1)/2\,,x) \\ \vdots & \ddots & \vdots \\ W_L(j,x) & \dots & W_L(j+N(R-1)/2\,,x) \end{bmatrix} \begin{bmatrix} I(j,x) \\ \vdots \\ I(j+N(R-1)/R\,,x) \end{bmatrix} \quad (1.\ 12)$$

It requires the condition of $L > R$ for solving eq.(1. 12), which means the number of coils is the upper bound of the number of reduction factor. To reconstruct MRI image from sensitivity-weighted aliased images, we can construct the sensitivity matrix W with a size of $L \times R$ in eq.(1. 12) based on the estimation of sensitivity profiles. On the other hand, the aliased pixels are collected from the inverse DFT of the acquired data. The pixels of MR image at the corresponding positions can be recovered by,

$$\mathrm{I} = (\mathrm{W}^H\mathrm{W})^{-1}\mathrm{W}^H\mathrm{a} \quad (1.\ 13)$$

This formulation is repeated for each pixel at (j, x), which represents the coordinate system of aliased spatial domain. Noise can be considered in this linear system, where generates an SNR-optimal results by the following,

$$\mathrm{I} = (\mathrm{W}^H\Lambda^{-1}\mathrm{W})^{-1}\mathrm{W}^H\Lambda^{-1}\mathrm{a} \quad (1.\ 14)$$

Here, Λ denotes the noise covariance matrix across all coils, which can be obtained by a noise-only preliminary scan without RF excitation. According to Papoulis generalized

sampling theorem, if sensitivity profiles are sufficient non-uniform, distinct, and is exact known, SENSE method provides the optimal reconstruction compare to SMASH, PILS, and GRAPPA. In addition, SENSE can be applied to 3D MRI, where the alias appears in two phase encoding direction, and thus it gains great reduction in scan time[19].

Non-Cartesian trajectories can also be combined with SENSE method. The difference is that the encoding matrix in eq.(1. 12) is not simply regarded as superposition. For example, the alias of the spiral trajectory is continuous and ring-shaped. Therefore, gridding algorithm is considered to construct encoding matrix[20]. The generalized form of SENSE method for arbitrary trajectories is formulated as

$$\begin{aligned}[W_1^H \quad W_2^H \quad \dots \quad W_L^H] \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_L \end{bmatrix} \mathrm{I} \\ = [W_1^H \quad W_2^H \quad \dots \quad W_L^H] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_L \end{bmatrix}\end{aligned} \tag{1. 15}$$

Encoding matrices W^H and W are generated by gridding algorithm, which can be improved by non-uniform fast Fourier transform (NUFFT)[21]. The desired image is reconstructed by solving eq.(1. 15) iteratively, such as conjugate gradient method.

Similar to SENSE, SPACE RIP solves a linear system of eq. (1. 7), where the Fourier transform has been taken along frequency encode direction (x-axis) preliminary after the acquired data is digitized. When the phase encoding gradient is applied alone, the acquired data is the Fourier-encoded projection of a sensitivity-weighted image onto x-axis. After digitized and applied IFFT, the eq.(1. 8) can be rewritten as

$$S_l\left(k_y{}^i, x\right) = \sum_{n=1}^{N} I(x,n) W_l(x,n) e^{jk_y{}^i n} \tag{1. 16}$$

For each m along the *x* direction, the matrix form of eq.(1. 9) becomes the following equation,

$$\begin{bmatrix} S_1\left(G_y{}^1, x\right) \\ S_1\left(G_y{}^2, x\right) \\ \vdots \\ S_1\left(G_y{}^N, x\right) \\ \vdots \\ \cdot \\ S_L\left(G_y{}^1, x\right) \\ \cdot \\ \cdot \\ S_L\left(G_y{}^N, x\right) \end{bmatrix}$$

$$= \begin{bmatrix} W_1(x,1)e^{jk_y{}^1 1} & \cdot & \cdot & W_1(x,N)e^{jk_y{}^1 N} \\ W_1(x,1)e^{jk_y{}^2 1} & \cdot & \cdot & W_1(x,N)e^{jk_y{}^2 N} \\ \vdots & \cdot & \cdot & \vdots \\ W_1(x,1)e^{jk_y{}^N 1} & \cdot & \cdot & W_1(x,N)e^{jk_y{}^N N} \\ \vdots & \vdots & \cdot & \vdots \\ \cdot & \cdot & \cdot & \cdot \\ W_L(x,1)e^{jk_y{}^1 1} & \cdot & \cdot & W_L(x,N)e^{jk_y{}^1 N} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ W_L(x,1)e^{jk_y{}^N 1} & \cdot & \cdot & W_L(x,N)e^{jk_y{}^N N} \end{bmatrix} \tag{1. 17}$$

$$\cdot \begin{bmatrix} I(x,1) \\ I(x,2) \\ \vdots \\ I(x,N) \end{bmatrix}$$

The vector of the received data on the left-hand side has $R \times L$ elements. The matrix on the right hand side is phase-encoded and sensitivity-weight. The vector I with a size of N represents one column of the desired image. Therefore, the whole image can be recovered

column by column independently according to different positions along x, which means that all columns can be computed simultaneously and can be speeded up by resorting to parallel computations. Similar to SENSE, it allows arbitrary down-sampling along phase encoding direction, but causes the linear system poor conditioning if $R > L$ or the noise errors of sensitivity estimation.

Both the reconstructions of SENSE and SPACE RIP techniques are based on the matrix inversion, where the matrix is a function of phase-encoding trajectories and the coefficients of sensitivity profiles. Therefore, carefully selecting k-space trajectories and minimizing the noise of estimated sensitivities play important roles in the performance of SENSE and SPACE RIP methods. Some groups adopted iterative regularizations for solving this inverse problem, such as POCS[22]. Tikhonov regularization is also commonly applied to SENSE method for eliminating noise effect or alias artifacts [23]. As the reduction factor increases, causing the data inconsistent, heavy regularization is usually applied. Bregman iteration is proposed to give sharp and better image structures compared to Tikhonov regularization[24]. On the other hand, the accuracy of sensitivity profiles cannot be guaranteed by estimating from a low-resolution image or via a separate scan, and this usually causes alias artifacts. To reduce the incorrect estimation of sensitivity profiles, JSENSE reconstructs MR images without prior knowledge of sensitivity[25]. Based on this linear system, it regards coil sensitivities and the desired image as unknown variables and uses an iterative optimization algorithm to solve the nonlinear problem. For the past decade, compressed sensing has emerged in the application of MR reconstruction. The concept of sparsity was applied in the linear system

of SENSE method, allowing total variation or l_1 norm regularization to further reduce the noise and alias artifacts. This will be discussed in the next chapter.

CHAPTER II

COMPRESSED SENSING MRI

Based on the conventional sampling theory, the bandlimited signals can be perfectly recovered without any alias when they are sampled above Nyquist rate, which is twice the bandwidth of the signals. In MRI scanning, sampling at Nyquist rate requires longer acquisition time and it causes patients' discomfort, motion blur, and higher medical expense. For the past few years, Compressed Sensing emerged as an innovative theory, which allows reducing measurements and recovering the signal by exploiting its sparsity and compressibility [26-31]. CS has been found to be favorable for applying in MR images because many MR images are compressible and sparse in the finite difference or wavelet transform domain. In addition, random sampling in the k-space provides incoherence of the measurements, which is highly needed for applying CS and is feasible in MR imaging. Therefore, the number of acquired MRI data can be significantly lower than the traditional sampling rate, and the images can be reconstructed and improved by optimization algorithms according to CS theory. To make CS applied in MRI successful, both the properties of sparsity and incoherence play important parts. The basic concepts of CS theory and its applications in accelerating MRI scanning will be introduced in this chapter.

Sparsity and Incoherence

The simple definition of "sparsity" is that the signal with a length N can be described by a basis, where there are only K nonzero coefficients and $K \ll N$. Many signals, including

natural images and most MR images, are compressible or "approximately" sparse in certain sparsifying transforms. For example, discrete cosine transform (DCT) or discrete wavelet transform (DWT) can map images into sparse coefficients, only few of which have large magnitudes and the others are zero or very small. This property makes signals compressible without much loss of information. DCT and DWT have been widely used in JPEG or MPEG for compressing natural images. As for MR images, unlike data compression after acquiring all the data, we exploit the sparsity at the beginning of data acquisition and recover the signals from fewer measurements, and this is CS all about.

Mathematically, any signal can be well described by a known basis. Considering a real-valued finite-length signal, which can be converted into a one-dimension vector, $\mathbf{x} \in \mathrm{R}^N$, now we can express the signal **x** as the following equation,

$$\mathbf{x} = \sum_{i=1}^{N} \mathrm{y}_i \psi_i \text{ or } \mathbf{x} = \mathbf{\Psi y} \qquad (2.\ 1)$$

where $\psi_i \in \mathrm{R}^N$. $\{\psi_i\}_{i=1}^{N}$ forming a $N \times N$ matrix $\mathbf{\Psi}$ can be a certain sparsifying transform. All weighting coefficients, $\{y_i\}_{i=1}^{N}$, measured by $\mathrm{y}_i = \langle \mathbf{x}, \psi_i \rangle = {\psi_i}^T \mathbf{x}$, form a N-length vector **y**. When just a few measurements can exactly represent the signal **x**, we say that the signal **x** is sparse. More specifically, the signal **x** is defined to be *K*-sparse if

$$\|\mathbf{y}\|_0 = \sum_{n=0}^{N-1} \mathbf{1}_{\{\mathbf{y}[\mathrm{n}] \neq 0\}} \leq K \qquad (2.\ 2)$$

Where the l_0 norm denotes the numbers of nonzero sparse coefficients of the signal, defined in eq.(2. 2). Because it is constant for $\mathbf{y}[\mathrm{n}] \neq 0$, the measure of l_0 norm makes a larger nonzero coefficient as significant as a smaller nonzero coefficient. The sparsity gives us an idea that a finite discrete signal can be exactly described by a number of

degrees of freedom, K, which is much smaller than its length. Under this sparsity assumption, we may want to reconstruct the signal $\mathbf{y}$ by solving the following optimization problem,

$$\begin{aligned} &\min \|\mathbf{y}\|_0 \\ &\text{s.t.}\, \mathbf{x} = \mathbf{\Psi y} \end{aligned} \tag{2. 3}$$

However, l_0 norm is not a convex function. Also it is discontinuous at the origin. The optimization problem become a combinatorial search and is usually intractable to solve. A most common alternative is the l_1 norm, which is defined as $\|\mathbf{y}\|_1 = \sum_{n=0}^{N-1}|\mathbf{y}[\mathrm{n}]|$. The measure of l_1 norm satisfies the scalability and the triangle inequality of norm properties and is most importantly convex. Therefore, the above combinatorial problem in eq.(2. 3) can be recast as a linear program by replacing the objective function with the measure of l_1 norm. Since it is a convex problem, there is a wide variety of techniques for solving l_1 minimization quite efficiently. The other regularizations are popular used, such as the l_p ($0 < p < 1$) norms and the widely-used l_2 norm, which are defined as $\|\mathbf{y}\|_p^p = \sum_{n=0}^{N-1}|\mathbf{y}[\mathrm{n}]|^p$, $\|\mathbf{y}\|_2^2 = \sum_{n=0}^{N-1}|\mathbf{y}[\mathrm{n}]|^2$ respectively. As shown in Figure 2.1, the balls for the l_0, l_p, l_1, and l_2 penalty functions are depicted in R^2 space and the blue line represents the feasible set of $\mathbf{x} = \mathbf{\Psi y}$, which will be a hyperplane in R^N space. As minimizing eq.(2. 3) with respective to different norm of penalty functions, we blow up the ball approaching to the line. Assume that $\mathbf{y}_0$ is the sparse solution close to the coordinate axes and we try to recover. As shown in Figure 2.1(a)(b)(c), the l_0, l_p and l_1 norms provide higher chance to find the true solution since the ball may intersect the line at $\mathbf{y}_0$. However, the l_0 and l_p

norms are not convex (the line may intersect the ball at multiple points), while l_1 norm makes the problem convex and can be recast as a linear program. On the other hand, l_2 ball touches the line at $\mathbf{y}_1$, which is close to the origin but not sparse, compared to the true solution, $\mathbf{y}_0$. Therefore, we usually resort to l_1 minimization to recover the sparse solution with higher accuracy.

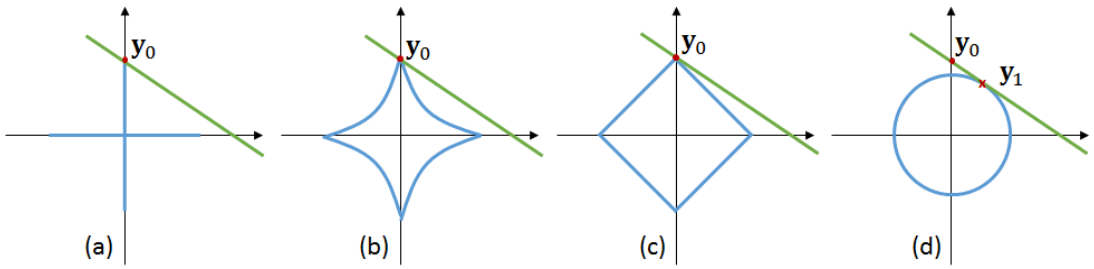


Figure 2.1 Minimizations of a sparse signal using different norms; the green line represents the feasible set, $\mathbf{x} = \Psi\mathbf{y}$, and the balls in blue represent (a) l_0 norm (b) $l_p(0 < p < 1)$ norm (c) l_1 norm (d) l_2 norm.

Another important requirement for using CS is incoherence, which is about the property of sensing matrix. That means the signal, which has a sparse representation in the sparse domain, $\mathbf{\Psi}$, must spread out in the domain where it is acquired. For example, a delta function in time domain is constant in frequency domain, which is so-called "spread out". Then, most important of all, with "random" undersampling in the domain where it is acquired, the artifacts are incoherent, which means the signal is noise-like in the domain of sparsifying transform. More specifically, the mathematical form of the coherence is defined as the following equation,

$$\mu(\mathrm{P},\Phi) = \sqrt{N} \cdot \max_{1\le k,j\le n} \left|\langle p_k, \phi_j\rangle\right|$$

The quantity of μ measures the maximum correlation between two elements of P and Φ. If the elements of P and Φ are correlated, the quantity of coherence μ is larger than 1. It menas that P and Φ is incoherent when $\mu = 1$, and they are hightly correlated when $\mu = \sqrt{N}$. For example, Assume that P represents the impulse function $p_k(t) = \delta(t-k)$, and Φ represents the Fourier transform, where $\phi_j = \frac{1}{\sqrt{N}} e^{-i2\pi jt/N}$. Because the Fourier transform of an impulse fuction is a constant, it is obvious that they are incoherent. From the quantity point of view, $\mu(\mathrm{P},\Phi) = 1$ is the minimum value of coherence, which means incoherence. Another example, commonly used in MRI application, is that P represents the randomly-sensing matrix, and Φ is the Fourier transform of MRI system. It has been proven that randomly-sampling matrices are considered incoherent with other fixed transform, such as Fourier or wavelet. Based on the compressed sensing theory, for instance, the matrix element p_k is choosen as independent identically distributed random variables according to Gaussian or $\pm$binary. Random matrices P and any fixed basis Φ can be shown to have restricted isometry property(RIP) with high probability, which alternatively means that the randomly-sensing matrix P has a low coherence with every possible fixed Φ. These two important properties, sparsity and incoherence lay the foundation of MRI using compressed sensing.

Compressed Sensing MRI and Reconstruction Algorithms

Compressed sensing has been widely applied to the data acquisition of MRI because it

provides great potential for reducing data samples, which can shorten the acquisition time. In [32], wavelet transform together with total variation were first used in sparsifying the representations of MR images in that most MR images are compressible. In addition, although random sampling in all directions of k-space is impractical, Cartesian non-uniform undersampling in phase encoding direction is still controllable by precisely changing the gradient encode during the acquisition, which makes the alias artifacts incoherent (noise-like). This Cartesian non-uniform sampling method can be applied in three applications of MRI data acquisition: two-dimensional single slices, two-dimensional multiple slices, and three-dimensional angiogram. Their sampling k-space trajectories are shown in Figure 2.2 respectively.

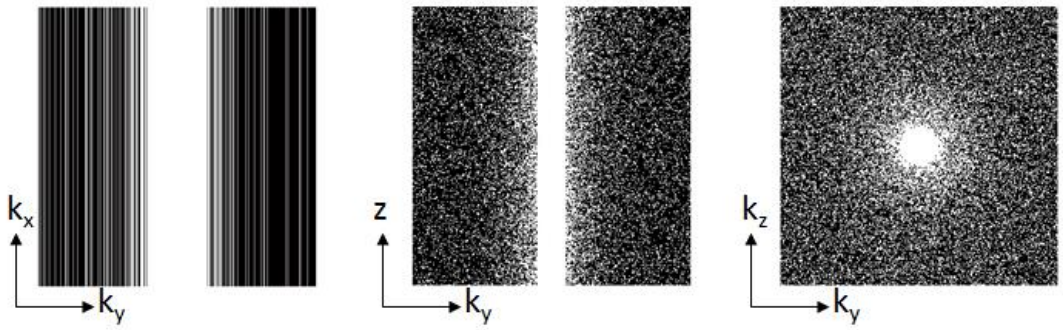


Figure 2.2 Random sampling for (a)2D single slice (b)2D multi-slice (c)3D angiogram [79].

In the first application, only one dimension can be subsampled along phase encoding direction, where frequency encoding has to be kept smoothly due to the limitation of MRI hardware consideration. Therefore, the reconstruction is modest in this

application. In the second application, the different random sampling trajectories are applied to different slices, so that it provides lower coherence and the sparsity along the slice direction can be exploited if the slices are thin and have some spatial redundancy. In the application of 3D angiogram, it allows random under-sampling along the k_y-k_z plane, which contributes to a higher degree of incoherence. The reduction of scan time in 3D case is more demanding than 2D case and CS MRI provides an attractive method in 3D angiogram applications. In addition, spatial-temporal sparsity is also exploited in dynamic MRI sequences, such as cardiac imaging [33, 34]. By randomly sampling along the phase encoding direction (k_y), the alias is incoherent on the plane of spatial temporal-frequency space (y-f), where the Fourier transform along the temporal direction generates sparse and periodic representations.

Based on compressed sensing theory, the way to reconstruct MR images from the incompletely sampled data is trying to enforce the image sparsity. The formulation of MRI data acquisition is commonly modeled as follows,

$$\mathbf{v} = P\Phi\mathbf{u} + \mathbf{n} \tag{2. 4}$$

Where P represents the randomly-sensing matrix, and Φ is the Fourier transform of MRI system. $\mathbf{u} \in C^N$ is the MR images to be reconstructed. $\mathbf{v} \in C^M$ is the acquired k-space measurements with the noise $\mathbf{n}$, where M is in the order of K. Asume that Ψ is the wavelet transform and $\mathbf{y}$ is the sparse representation of wavelet transfom, then the model becomes the following equation, $\mathbf{v} = \mathrm{P}\Phi\Psi\mathbf{y} + \mathbf{n}$, where $\mathbf{u} = \Psi\mathbf{y}$. Instead of solving l_0 norm problem, l_1 norm minimization is more tractable and computationally feasible not only because it provides a high chance to find the solutions as same as to the l_p minimization

problem($p < 1$) with the explainations of Figure 2.1, but also there exist a wide variety of efficient numerical solvers using convex optimization. Therefore, the unknown image $\mathbf{u}$ can be reconstructed by minimizing the following l_1 norm problem, which is equal to solving a constrained convex optimization.

$$\begin{aligned} &\min \|\mathbf{y}\|_1 \\ &\text{s.t.} \|\mathrm{P}\Phi\mathbf{u} - \mathbf{v}\|_2 < \varepsilon \end{aligned} \tag{2. 5}$$

The constraint is referred to the distance between the measured data and the estimated data, which is commonly defined by l_2 norm. Here, ε is the magnitude of errors, usually set according to the variance of noise or the approximation of allowable errors. The objective function can promote the sparsity in wavelet domain, and the constraint enforces the consistency to the measured data and is regarded as a regularization term.

In addition, finite differences of the image, which is well known as total variation (*TV*), can also be regarded as a sparsifying transform. In sparse MRI, it is defined by the sum of absolute variations of the image ($\sum_j |D_j \mathbf{u}|$, viewed as l_1 norm of image variations), which is well known as an anisotropic version of total variation. The summation with index j is taken over all pixels, ranging from 1 to N, and $D_j \mathbf{u} = \left[D_j^{(1)}; D_j^{(2)}\right]\mathbf{u}$ represents the vertical and horizontal finite differences of $\mathbf{u}$ at pixel j. Therefore, the objective function can be rewritten as exploiting the sparsity of wavelet representations and image variations.

$$\begin{aligned} &\min \|\mathbf{y}\|_1 + \alpha TV(\mathbf{u}) \\ &\text{s.t.} \|\mathrm{P}\Phi\mathbf{u} - \mathbf{v}\|_2 < \varepsilon \end{aligned} \tag{2. 6}$$

Where α controls the effect between the sparsity of wavelet representations and the finite differences. Eq.(2. 7) can be recast in a Lagrangian form and become an unconstrained convex optimization problem.

$$\arg\min_{u} \lambda \left\| \Psi^{-1}\mathbf{u} \right\|_1 + \alpha TV(\mathbf{u}) + \frac{1}{2}\|\mathrm{P}\Phi\mathbf{u} - \mathbf{v}\|_2^2 \qquad (2.\ 7)$$

Here, λ controls the penalty of data fidelity constraint, and governs the tradeoff between the sparsity. The larger the value of parameters λ and α, the more the solutions tend to be sparse. The l_2 norm tends to suppress the large coefficients, making the solution closer to the origin. Therefore, the estimated solutions may deviate from the true sparse solutions as the equation is heavily penalized by the term of l_2 norm regularization. Instead, when λ is chosen to emphasize the effect of minimizing l_1 norm, the estimated solution is usually sparse. For a proper choice of λ [35, 36], minimizing Eq.(2. 7) will yield the same recovery of Eq.(2. 6).

With Nyquist-Shannon sampling theory, the signal is linearly recovered via the interpolation of the Sinc function. However, unlike conventional signal processing, the signal recovery of CS is achieved by certain nonlinear methods[37, 38]. There are a wide variety of recovery algorithms for solving Eq.(2. 6). Among these advanced numerical techniques, greedy pursuit[39] and convex optimization[40, 41] are often computationally practical. The former relies on identifying the signal representations, which give the best quality improvement, and then iteratively refining the coefficients. The representative algorithms include orthogonal matching pursuit (OMP)[42] and iterative hard thresholding (IHT). A tremendous variety of algorithms in the field of convex

optimization is powerful for computing sparse representations. Iterative shrinkage-thresholding [43-45], which is related to gradient-descent methods, has been extensively used for past few years. The algorithms, such as interior point [46, 47], augmented Lagrangian method[48], alternating direction method of multipliers[49], Bregman iterative method [50-52], projection onto convex sets, and proximal gradient [53] are well-developed for solving the l_1 norm problem and CS reconstructions.

The iterative shrinkage, also known as soft-thresholding is widely used to solve the l_1 minimization problem. The general form is as follows,

$$\min_x \{F(x) \equiv G(x) + H(x)\} \tag{2. 8}$$

Here, $H(u)$ is a convex and differentiable function, such as l_2 norm function; $G(u)$ is a convex and nonsmooth function, such as l_1 norm function. More specifically, for a general l_1 regularization, the equation becomes,

$$\min_x \mu\|x\|_1 + H(x) \tag{2. 9}$$

A basic approximation model for eq.(2. 8) is written as

$$\min_x \left\{ Q_L(x,z) \equiv H(z) + \langle x - z, \nabla H(z) \rangle + \frac{L}{2}\|x - z\|^2 + G(x) \right\} \tag{2. 10}$$

For any $L > 0$, and a given z, the first and second terms of $Q_L(x,z)$ are the first order Taylor expansion of function $H(x)$ at z. Because the approximation is only accurate for z close to x, l_2 norm is added to the objective function as a penalty term. As a result, $\min_x F(x)$ implies $\min_x Q_L(x,z)$. Based on an iterative scheme, minimizing $Q_L(x,z)$ can

be rewritten as $\min_x G(x) + \frac{L}{2}\left\|x - (z - \frac{1}{L}\nabla H(z))\right\|^2$ because the two objective functions are different only by a constant and the derivations are as follows,

$$\min_x Q_L(x,z) \equiv \min_x \left\{H(z) + \langle x - z, \nabla H(z)\rangle + \frac{L}{2}\|x - z\|^2 + G(x)\right\}$$

$$= \min_x \left\{H(z) + (x - z)^T \nabla H(z) + \frac{L}{2}(x - z)^T(x - z) + G(x)\right\}$$

$$= \min_x \left\{H(z) + x^T \nabla H(z) - z^T \nabla H(z) + \frac{L}{2}(x^T x - x^T z - z^T x + z^T z) + G(x)\right\}$$

$$= \min_x \left\{x^T \nabla H(z) + \frac{L}{2}(x^T x - x^T z - z^T x) + G(x)\right\}$$

$$= \min_x \left\{\frac{L}{2}\left[x^T x - x^T\left(z - \frac{1}{L}\nabla H(z)\right) - \left(z - \frac{1}{L}\nabla H(z)\right)^T x\right] + G(x)\right\}$$

$$= \min_x \left\{\frac{L}{2}\left[x^T x - x^T\left(z - \frac{1}{L}\nabla H(z)\right) - \left(z - \frac{1}{L}\nabla H(z)\right)^T x + \left(z - \frac{1}{L}\nabla H(z)\right)^T\left(z - \frac{1}{L}\nabla H(z)\right)\right] + G(x)\right\}$$

$$= \min_x \left\{\frac{L}{2}\left[x - \left(z - \frac{1}{L}\nabla H(z)\right)\right]^T\left[x - \left(z - \frac{1}{L}\nabla H(z)\right)\right] + G(x)\right\}$$

$$= \min_x \left\{\frac{L}{2}\left\|x - \left(z - \frac{1}{L}\nabla H(z)\right)\right\|^2 + G(x)\right\}$$

Assume that $p_L(z) = \arg\min_x Q_L(x,z)$, the iterative algorithm performs in a form of $x^{k+1} = p_L(x^k)$. For l_1 norm regularization, the iterative algorithm becomes the following equation.

$$x^{k+1} \leftarrow \arg\min_x \mu\|x\|_1 + \frac{1}{2\delta^k}\|x - (x^k - \delta^k \nabla H(x^k))\|^2 = \arg\min_x f(x) \quad (2.\ 11)$$

For $k = 0,1, \ldots$, start from x^0. Here, the parameter δ^k represents the step size and is set to positive. Note that the unknown variable x is component-wise separable, which means that each component $x_i (i = 0,1, \ldots, n)$ of a vector x can be independent obtained by the shrinkage operator, also referred to as soft-thresholding shown in Figure 2.3,

$$x_i^{k+1} = \text{shrink}(\left(x^k - \delta^k \nabla H(x^k)\right)_i, \mu\delta^k) \quad (2.\ 12)$$

where the shrinkage function is defined as follows.

$$\text{shrink}(y, \alpha) \coloneqq \text{sign}(y)\max\{|y| - \alpha, 0\} = \begin{cases} y - \alpha & y \in (\alpha, \infty) \\ 0 & y \in [-\alpha, \alpha] \\ y + \alpha & y \in (-\alpha, -\infty) \end{cases} \quad (2.\ 13)$$

The original shrinkage function is shown in Figure 2.3(a). Eq.(2. 11) yields the result of eq.(2. 12) by taking the derivative of $f(x)$ and setting to zero. For each entry, $\frac{\partial f}{\partial x_i} = 0$ leads to the following equations.

$$\mu \cdot \text{sign}(x_i^*) + \frac{1}{\delta^k}(x_i^* - \left(x^k - \delta^k \nabla H(x^k)\right)_i) = 0$$

$$x_i^* + \mu\delta^k \cdot \text{sign}(x_i^*) = \left(x^k - \delta^k \nabla H(x^k)\right)_i \quad (2.\ 14)$$

Take the absolute value on both sides.

$$
\begin{aligned}
|x_i^* + \mu\delta^k \cdot \operatorname{sign}(x_i^*)| &= \left|\left(x^k - \delta^k \nabla H(x^k)\right)_i\right| \\
&= |x_i^*| + \mu\delta^k
\end{aligned}
\tag{2. 15}
$$

From eq.(2. 14),

$$
\begin{aligned}
x_i^* &= \left(x^k - \delta^k \nabla H(x^k)\right)_i - \mu\delta^k \cdot \operatorname{sign}(x_i^*) \\
&= \left(x^k - \delta^k \nabla H(x^k)\right)_i - \mu\delta^k \\
&\quad \cdot \operatorname{sign}\left(\left(x^k - \delta^k \nabla H(x^k)\right)_i\right)
\end{aligned}
\tag{2. 16}
$$

where

$$
\operatorname{sign}(x_i^*) = \frac{\operatorname{sign}(x_i^*)(|x_i^*| + \mu\delta^k)}{|x_i^*| + \mu\delta^k} = \frac{\operatorname{sign}(x_i^*)|x_i^*| + \operatorname{sign}(x_i^*)\mu\delta^k}{|x_i^*| + \mu\delta^k} = \frac{x_i^* + \operatorname{sign}(x_i^*)\mu\delta^k}{|x_i^*| + \mu\delta^k}
$$

$$
= \frac{\left(x^k - \delta^k \nabla H(x^k)\right)_i}{\left|\left(x^k - \delta^k \nabla H(x^k)\right)_i\right|} = \operatorname{sign}(\left(x^k - \delta^k \nabla H(x^k)\right)_i)
$$

Let $x_i^{k+1} = x_i^*$, then eq.(2. 14) becomes the iterative algorithm for l_1 norm regularization, and are illustrated in Figure 2.3(b), which is so-called shrinkage function.

$$
\begin{aligned}
x_i^{k+1} = x_i^* &= \operatorname{sign}\left(\left(x^k - \delta^k \nabla H(x^k)\right)_i\right) \\
&\quad \cdot \max\left\{\left|\left(x^k - \delta^k \nabla H(x^k)\right)_i\right|\right. \\
&\quad \left. - \mu\delta^k, 0\right\} \\
&= \operatorname{shrink}(\left(x^k\right. \\
&\quad \left. - \delta^k \nabla H(x^k)\right)_i, \mu\delta^k)
\end{aligned}
\tag{2. 17}
$$

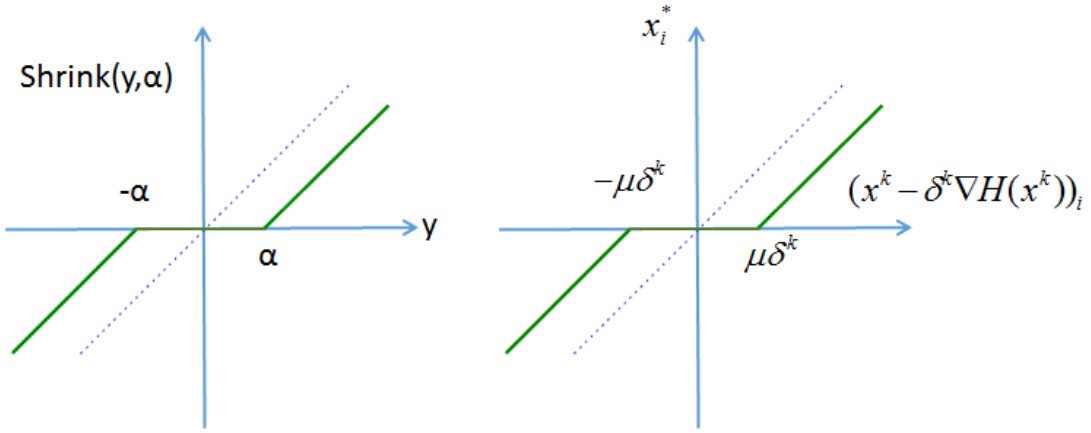


Figure 2.3 Shrinkage function (a) of the original and (b) applied in l_1 norm regularization.

The shrinkage method can also be applied to *TV* regularization by replacing the term of l_1 norm with *TV* function, leading to the following equation.

$$\min_x \mu TV(x) + H(x)$$

$$x^{k+1} \leftarrow \arg\min_x \mu TV(x) + \frac{1}{2\delta^k}\|x - (x^k - \delta^k \nabla H(x^k))\|^2$$

With the same iterative procedure, the solution to eq.(2. 6) can be efficiently found by solving the two sub-problem.

Another widely used algorithm, known as Alternating direction method of multipliers (ADMM), has drawn great attention for past decades due to the needs of solving large-scale statistical tasks. ADMM decomposes a massive optimization problem into distributed convex optimizations, which can be easily implemented by parallel computing with a distributed-memory. Considering a general form of convex problem,

$$\min_x F(x) + H(Ax) \tag{2. 18}$$

Here, $x \in R^n$, $F(x)$ and $H(Ax)$ are convex functions, and A is $m \times n$ matrix. The problem can be rewritten as a constrained minimization problem by introducing an additional variable, $y \in R^m$,

$$\begin{gathered} \min_x F(x) + H(y) \\ \text{s.t. } Ax = y \end{gathered} \tag{2. 19}$$

The augmented Lagrangian algorithm for problem (2. 19) is

$$\begin{aligned} L_\rho(x, z) = F(x) + H(y) + z^T(Ax - y) \\ + \frac{\rho}{2}\|Ax - y\|^2 \end{aligned} \tag{2. 20}$$

Here, $z \in R^m$ is the dual variable or Lagrangian multiplier and $\rho > 0$ is known as penalty parameter, where it is equivalent to standard Lagrangian form when $\rho = 0$. By applying dual ascent method, the recursions consist of the following steps.

$$\begin{gathered} (x^{k+1}, y^{k+1}) \in \arg\min_{x,y} L_\rho\,(x, y, z^k) \\ z^{k+1} = z^k + \rho(Ax^{k+1} - y^{k+1}) \end{gathered} \tag{2. 21}$$

This algorithm is known as the method of multipliers. The two primal variables, x and y is jointly updated by minimizing the augmented Lagrangian, and the dual variable (Lagrangian multiplier), z is estimated by the constraint. The method of multipliers becomes less attractive because minimizing augmented Lagrangian makes the separable objective functions, $F(x)$ and $H(y)$, be strongly coupled together by the penalty term, $\frac{\rho}{2}\|Ax - y\|^2$. On the contrary, ADMM decouples the objective function and updates the

primal variables in an alternating fashion, which denotes the name "alternating direction", as shown in the following iterations.

$$(x^{k+1}) \coloneqq \arg\min_{x} L_{\rho}\,(x, y^{k}, z^{k})$$

$$(y^{k+1}) \coloneqq \arg\min_{y} L_{\rho}\,(x^{k+1}, y, z^{k}) \qquad (2.\ 22)$$

$$z^{k+1} = z^{k} + \rho(Ax^{k+1} - y^{k+1})$$

The algorithm consists of three steps. The tangled minimization problem is divided into two sub-problems. In the first step, the minimization focuses on a quadratic perturbation of function $F(x)$ with respect to the variable x while y^k and z^k are fixed. In the second step, the minimization focuses on a quadratic perturbation of function $H(y)$ with respect to the variable y while x^{k+1} and z^k are fixed as constant. With the final step, the update of dual variable z uses the penalty parameter ρ as a step size. The decoupling minimization steps allow exploiting individual convex property of objective functions, so that it is possible to compute in an efficient manner or carry out in parallel computing. That's how ADMM can take advantage of the decomposability of dual ascent and the augmented Lagrangian for constrained optimization.

According to a recent research, the well-known Bregman iterative algorithm, is equivalent to the method of multipliers, and the split Bregman method, which has been applied in solving l_1 minimization problem and CS applications, is actually equivalent to ADMM. Although theoretical results of these algorithms were established few decades ago, they are being used more widely as the needs for analyzing large-scale data are growing and the massive computing system becomes available.

Compressed Sensing MRI with Multiple Coils

Since imaging speed has been a major part of MRI revolutions and becomes the limitation of MRI applications, up-to-date MRI scanners are often equipped with the phased array receiver system, which has multiple coils as RF receiver for parallel collecting data. As a result, various combinations of parallel imaging and compressed sensing or sparsity have developed for further speeding up the scan time.

Compressed sensing was first tried to combine with SENSE parallel imaging method [24, 54-57]. SparseSENSE is one of these techniques directly extending sparseMRI to SENSE framework. The reconstruction is achieved by minimizing the following equation

$$\begin{aligned} \hat{\mathbf{u}} &= \min_{\mathbf{u}} \|\Psi \mathbf{u}\|_1 + \alpha TV(\mathbf{u}) \\ &\text{s.t.} \|F_u S\mathbf{u} - \mathbf{v}\|_2 < \varepsilon \end{aligned} \tag{2. 23}$$

Here, $F_u S$ is defined as a under-sampled Fourier and sensitivity-encoding matrix. As the reduction factor is practically smaller than the number of channels, $F_u S\mathbf{u}$ represents an overdetermined system. However, CS primarily considers solving an underdetermined system and needs Φ to be an orthonormal basis. In addition, the incoherence between the random sampling matrix and the sensitivity encoding matrix, which can make system ill-conditioned due to the inaccuracy of estimating sensitivity, has not been explored. Therefore, sparseSENSE is still regarded as an l_1-regularized SENSE using sparsity.

In CS-SENSE method [58], it divides the reconstruction into two stages. The first step is to clean up the alias of coil images channel by channel, where random sampling causes the eliminated alias.

$$\widehat{\mathbf{u}_l^A} = \min_{\mathbf{u}_l^A} \left\| \Psi \mathbf{u}_l^A \right\|_1 + \alpha TV(\mathbf{u}_l^A) \quad \text{s.t.} \left\| F_u \mathbf{u}_l^A - \mathbf{v}_l \right\|_2 < \varepsilon \tag{2. 24}$$

Here, $\mathbf{u}_l^A$ is the aliased coil image from the l^{th} channel with a reduced FOV. The encoding matrix Φ solely represents the Fourier transform since the coil images $\mathbf{u}_l^A$ are modulated by coil sensitivity. In the second step, the aliased coil images are combined pixel-by-pixel using the Cartesian SENSE method. The performance of these methods rely on the accuracy of measuring coil sensitivities, where the estimation can be improved by distributed compressed sensing[59, 60].

In Self-feeding sparse SENSE[61], it improved the performance of sparseSENSE by splitting eq.(2. 24) with auxiliary variables as follows.

$$\min_{\mathbf{x}} \sum_{l=1}^{c} \|F_u S_l \mathbf{u} - \mathbf{v}_l\|_2 + \lambda \|\Psi \hat{\mathbf{u}}\|_1 + \alpha \|\nabla \hat{\mathbf{u}}\|_1 \quad \text{s.t.} \ \hat{\mathbf{u}} = \mathbf{u} \tag{2. 25}$$

Here, $\|\nabla \hat{\mathbf{u}}\|_1$is l_1 norm of finite difference of the desired image. The problem can turn to un-constrained minimization by placing a quadratic function and penalizing it with a sufficient large μ.

$$\min_{\mathbf{x}} \sum_{l=1}^{c} \|F_u S_l \mathbf{u} - \mathbf{v}_l\|_2 + \lambda \|\Psi \hat{\mathbf{u}}\|_1 + \alpha \|\nabla \hat{\mathbf{u}}\|_1 + \mu^2 \|\hat{\mathbf{u}} - \mathbf{u}\|_2^2 \tag{2. 26}$$

Then, based on the alternating minimization method, the image can be iteratively reconstructed by splitting variables and alternatively solving the $\mathbf{u}$ sub-problem and the $\hat{\mathbf{u}}$ sub-problem.

$$\begin{aligned} &\min_{\mathbf{u}} \sum_{l=1}^{c} \|F_u S_l \mathbf{u} - \mathbf{v}_l\|_2 + \mu^2 \|\hat{\mathbf{u}} - \mathbf{u}\|_2^2 \\ &\min_{\hat{\mathbf{u}}} \mu^2 \|\hat{\mathbf{u}} - \mathbf{u}\|_2^2 + \lambda \|\Psi \hat{\mathbf{u}}\|_1 + \alpha \|\nabla \hat{\mathbf{u}}\|_1 \end{aligned} \tag{2. 27}$$

The $\mathbf{u}$ sub-problem is regarded as regularized SENSE with prior information $\hat{\mathbf{u}}$. The $\hat{\mathbf{u}}$ sub-problem is well known as image denoising with total variation and wavelet. It has no effect of setting $\mu^2 = 1$ because λ and α can balance three terms sufficiently. The map of g-factor is used to regularize the need of enforcing sparsity, and then $\hat{\mathbf{u}}$ subproblem becomes

$$\min_{\hat{\mathbf{u}}} \|\hat{\mathbf{u}} - \mathbf{u}\|_2^2 + \lambda \|(\mathrm{g} - 1)\Psi \hat{\mathbf{u}}\|_1 + \alpha \|(\mathrm{g} - 1)\nabla \hat{\mathbf{u}}\|_1 \tag{2. 28}$$

In addition, sensitivities can be iteratively updated after $\hat{\mathbf{u}}$ subproblem and the reconstructed image can be improved by a better estimation of sensitivities.

Sparse BLIP provides an iterative manner to simultaneously reconstruct sensitivities and the image by exploiting the sparsity of both sensitivities and the

image[62]. The image and sensitivities are jointly reconstructed by minimizing the following equation.

$$\min_{\mathbf{u}} \sum_{l=1}^{c} \|F_u(\mathrm{S}_l \cdot \mathbf{u}) - \mathbf{v}_l\|_2 + \lambda\|\Psi\mathbf{u}\|_1 + \alpha\|\nabla\mathbf{u}\|_1 + \beta \sum_{l=1}^{c} \|\nabla \mathrm{S}_l\|_1 \tag{2. 29}$$

Here, $(\cdot)$ denotes the component-wise product. Based on the alternating minimization method, the energy function can be split into two sub-problems with respective to $\mathbf{u}$ and S_l. With fixing $\mathrm{S}_l = {\mathrm{S}_l}^{(k-1)}$, the image is updated as follows

$$\mathbf{u}^k = \arg\min_{\mathbf{u}} \sum_{l=1}^{c} \left\|F_u({\mathrm{S}_l}^{(k-1)} \cdot \mathbf{u}) - \mathbf{v}_l\right\|_2 + \lambda\|\Psi\mathbf{u}\|_1 + \alpha\|\nabla\mathbf{u}\|_1 \tag{2. 30}$$

With fixing $\mathbf{u} = \mathbf{u}^k$, sensitivities are updated by minimizing the following equation.

$${\mathrm{S}_l}^k = \arg\min_{\mathrm{S}_l} \sum_{l=1}^{c} \|F_u(\mathrm{S}_l \cdot \mathbf{u}^k) - \mathbf{v}_l\|_2 + \beta \sum_{l=1}^{c} \|\nabla \mathrm{S}_l\|_1 \tag{2. 31}$$

Eq.(2. 30) is the same as sparse SENSE, where sparse BLIP improves sparse SENSE by iteratively alternating the reconstructions of the image and sensitivities.

Some research papers focus on combining CS with GRAPPA. With sufficient density around the center region, GRAPPA is used to fill out the location of missing data, where leads to an incoherent-aliased image, and then used CS to eliminate the alias [63].

In addition, l_1 SPIR-iT uses iterative Self-consistent Parallel Imaging reconstruction (SPIR-iT), which is an iterative GRAPPA-like approach, incorporating the sparsity into the auto calibrating system [64-66]. The problem is formulated as follows.

$$\begin{aligned} &\min \text{Joint} l_1(\Psi \mathbf{u}) \\ &\text{s.t.} G\mathbf{m} = \mathbf{m} \\ &F_u \mathbf{u} = \mathbf{v} \\ &F\mathbf{u} = \mathbf{m} \end{aligned} \tag{2. 32}$$

Here, **u** is the desired image, **v** is the acquired data, and **m** is the reconstructed k space data. F represents a full-sampled Fourier encoding matrix, and F_u represents Fourier encoding matrix with Poisson disc sampling, which can provide more incoherent artifacts. G is defined as an interpolation kernel, which is obtained from calibration. l_1 SPIR-iT used joint l_1 to enforcing sparsity, defined as $\sum_r \sqrt{\sum_c |w_{rc}|^2}$, where w_{rc} is the coefficient of sparse transform. With this definition, they penalize the coefficient from all coils but at same spatial location and exploit both the sparsity and data consistency.

The above approaches incorporating CS require l_1 minimization algorithms on large-scale problem, which has well developed since the late 1970's and early 1980's due to the explosion of computing power. That means the numerical algorithms are often computationally expensive. Especially for three-dimensional MRI acquisition from multiple coils, the high acceleration of scan time is achieved by combining parallel imaging and CS. However, the cost could be an excessively long time in image reconstructions, which make clinically impractical. For past decade, massive parallel computing, such as graphics processing units (GPU) or multicore central processing unit

(CPU) has dramatically evolved, providing a good tool for calculating high computation complexity operations on large-scale datasets. Multi-core processors draw great attentions in accelerating the runtime of CS MRI reconstructions. Therefore, this dissertation focuses on improving and speeding up the CS reconstructions of multichannel MRI data.

CHAPTER III

COMPRESSED SENSING MRI WITH MULTICHANNEL DATA USING MULTICORE PROCESSORS*

Objective

MRI using conventional Fourier imaging is relatively slow as compared with other imaging modalities because the data acquisition speed is limited by hardware capability and SNR factors. CS is a new sampling and reconstruction technique that allows a sparse signal to be reconstructed from a set of randomly undersampled data, which in turn shortens the data acquisition time [67-69]. Recently, Lustig et al have demonstrated the use of CS in a number of MRI applications with remarkable acceleration [32].

Because array systems receive signals from multi-channel simultaneously, they offer improved SNR [7, 70] or accelerated speed in PI methods. Several groups including ours have investigated methods to integrate CS and PI with array systems [54-56, 71-73]. In these methods, CS and PI are generally coupled in a large linear system, which contains data from all channels. Another novel approach is to apply the CS algorithm to reconstruct an aliased image in each channel, followed by a conventional SENSE method [58]. However, an alternative approach to integrate them is to apply the CS reconstruction to

---

each channel individually followed by a simple channel combination using a sum-of-squares method.

One significant problem in the aforementioned methods is the computational complexity. CS reconstruction involves non-linear optimization, which can be time-consuming even for regular-sized data and images. With multi-channel array systems, the computational complexity will scale with the number of channels. This problem can be partially addressed with improved performance of central processing units (CPU). Unfortunately, computation power growth using higher CPU clock frequency is limited by the power consumption and physical wire size. Modern processor design is moving towards multi-core architecture. The ubiquitously available duo-core, quad-core CPU, and Graphics Processing Unit (GPU) offer new platforms for implementing parallel algorithms to accelerate image reconstructions. In [74], a conjugate gradient (CG) solver was implemented on NVIDIA's G80 GPU. Borghi et al used the multi-core platform to solve the CS reconstruction which involves l_1 minimization [75]. Other advanced MRI reconstruction algorithms were also implemented on NVIDIA's Quadro FX 5600 GPU [76, 77]. These previous work shows the tremendous potential of multi-core processors for speeding up MRI reconstruction. However, most of the work is for single-channel data. In addition, they usually require significant efforts to program the parallel algorithms on the GPUs. For the multi-channel parallel imaging with CS algorithm, the methods in [54-56, 71-73] cannot be parallelized straightforwardly as the linear system is coupled. However, the methods discussed in [74-77] can be potentially adopted to accelerate these reconstructions. In contrast to the previous cases, the method in [58] can directly benefit

from parallel computing because processing of different channels is decoupled before using the SENSE reconstruction.

In this paper, we proposed an accelerated reconstruction procedure for combining CS with the array receive systems, using widely available multi-core CPUs to accelerate the image reconstruction. The results show that the reconstruction algorithm can benefit significantly from the parallel computing and multi-core architecture. This work is developed from a preliminary conference report in [78].

Methods

Based on the CS theory, an image **x** can be reconstructed from a reduced set of incomplete k-space data **y**, where $\mathbf{y} = \mathbf{\Phi x}$ with $\mathbf{\Phi}$ being the randomly-sampled Fourier transform operator implemented by encoding gradients. Specifically, the image **x** can be recovered by the following constrained optimization problem

$$\begin{aligned} &\min\ (\|\mathbf{\Psi x}\|_1 + \alpha\, \mathrm{TV}(\mathbf{x})\,) \\ &\mathrm{s.t.}\ \|\mathbf{y} - \mathbf{\Phi x}\|_2 < \varepsilon \end{aligned} \tag{3. 1}$$

Here $\mathbf{\Psi}$ is a linear transform such as the wavelet transform and TV(**x**) is the total variation of the image. Both terms reflect the sparsity of the image. Note that parameter α is to balance the two terms and parameter ε corresponds to the data fidelity, which can be set to zero for simplicity. With an array receiver system, a k-space data set will be acquired from each channel. An image $\mathbf{x}_k$ can be reconstructed from the *k*th channel data using the CS method. Then, the images from all channels can be combined using the well-known

sum-of-squares method [7]. A desirable feature of this approach is that individual channels are not coupled, which allows parallel reconstruction using multi-core architecture.

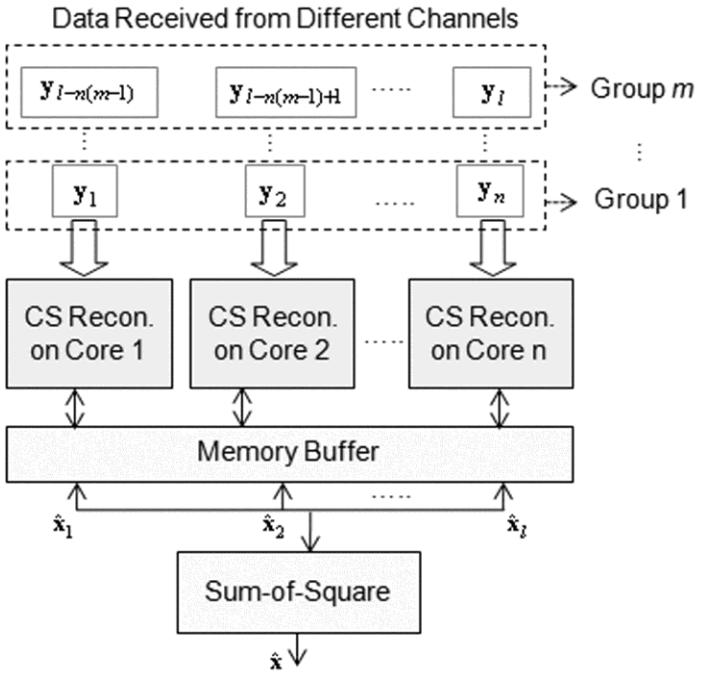


Figure 3.1 Illustration of the proposed algorithm on a multi-core CPU. Under-sampled multi-channel data are input to n cores of CPU. The final image is obtained by combining all individual channel images [85].

The overall reconstruction algorithm is illustrated in Figure 3.1. Here, $\mathbf{y}_k$, k = 1, 2, …, l, are the under-sampled data received from the l parallel channels. The data sets $\mathbf{y}_k$ are clustered into m groups. Each of the first m-1 groups contains n data sets, and the last group m contains the rest l-n(m-1) data sets. Note that n corresponds to the number of CPU cores for which is intended to be used. In reconstruction, m groups will be processed sequentially, i.e., the first group of n data sets is fed to the n available processing cores

simultaneously, followed by the second group, and so on. After all *m* groups are processed and all *l* individual channel images are reconstructed, the final image is obtained by the sum-of-squares combination. In each processing core, the reconstruction for the *k*th channel data is obtained by recasting Eq.(3. 1) as

$$\min \|\boldsymbol{\Phi}\hat{\mathbf{x}}_k - \mathbf{y}_k\|_2^2 + \lambda_1\|\boldsymbol{\Psi}\hat{\mathbf{x}}_k\|_1 + \lambda_2 \mathrm{TV}(\hat{\mathbf{x}}_k) \qquad (3.\ 2)$$
$$k = 1,2,\dots,l$$

In this paper, SparseMRI software [79] was used for optimization, whose algorithms are based on a non-linear CG method. The regularization parameters λ_1 and λ_2 are set to 0 and 0.001 experimentally.

The algorithm was tested on a platform that contained a standard Intel Core 2 Quad Q8200 2.33 GHz CPU, with 4 MB L2 cache and 4GB DDR2 memory. All computer simulation and image reconstruction were performed in Matlab (Mathworks, Natick, MA, built 7.7.0.471). In the default mode, there is no multithread computation supporting of the Parallel Computing Toolbox, a Matlab process is only able to utilize 25% of quad-core CPU. To initialize the parallel process as shown in Figure 3.1, a method similar to the message passing interface in [80] was used, which allows multiple Matlab processes run on parallel computer clusters or multi-cores. In the proposed method with quad-core CPU, four instances of Matlab are started at the beginning, each initiating a process. A primary process in one instance will save the k-space data and reconstruction parameters for each channel in a separate data file, and, in the end, will combine all channel images. As soon as the files are ready, the primary and the other three slave processes will simultaneously

access and reconstruct the channel data. This parallel processing will finish until all channel files are accessed and processed. At this point, the primary process, after realizing that there is no more individual channel data to be processed, will perform the sum-of-squares combination. To simplify programming and boost the efficiency, all processes communicate through file reading/writing (R/W) only. This also avoids the potential conflicts between processes and ensures that different processes do not access the same channel data.

To test the proposed algorithm, both simulated and in-vivo data were used. The multiple-channel (4-channel, 8-channel, and 16-channel) k-space data were simulated using the 'Shepp-Logan' phantom and Gaussian channel sensitivities. For the 4-channel data, five data sets with different size (32×32, 64×64, 128×128, 256×256, and 512×512) were synthesized. This was to test the effect of image size on saving the computation time and the overhead time. The individual channel data were under-sampled in the k-space with radial sampling pattern. The under-sampling factor was about 40%, which meant that only 40% of the total data were kept. For the 8-channel and 16-channel simulations, only 256×256 images were used with and an under-sampling factor of 33%. In addition, for simulated data, using 1, 2, 3, or 4 cores in the CPU generated four sets of results. The total reconstruction time was recorded and compared. The total time includes time for launching and communicating through file R/W and computation starvation, which is defined as the minimum absolute time that the primary or slaves waits for the data before continuing on the next operations. Note that if multiple cores were working simultaneously, we only recorded the maximum stalled time and the maximum overhead

time on file R/W among different cores of CPU. In addition, the computation speedup factor was evaluated by comparing the proposed method with the reconstruction time without the proposed method.

Finally, an 8-channel in-vivo brain MR data set was acquired and tested. The k-space data with a size of 256×256 from each channel were acquired in full field-of-view, i.e., without under-sampling. Then, the radial sampling was simulated by decimation with an under-sampling factor of 33%. The under-sampled 8-channel data were then input to the quad-core processor for image reconstruction as shown in Figure 3.1. To evaluate the efficiency of using multi-core parallel reconstruction, the total reconstruction times for both simulated data and in-vivo data were compared with the time required using the conventional reconstruction method, where the quad-core CPU is used without the proposed parallelization. To reduce the performance variation, each reconstruction test was repeated ten times and the average time cost was calculated and shown.

Results

Table 3.1(a) lists the total reconstruction time for all experiments with the simulated 4-channel data, including both time cost for file R/W and the CS reconstruction. As shown, using 2 cores leads to significant reduction in the computation time per core, with the reduction factor of about 1.7. In addition, using 4 cores gives the shortest reconstruction time, where the average reduction factor is about 2.4. It is worth mentioning that for the simulated 4-channel data, there is very little benefit from using 3 cores compared to using 2 cores. This is because, in both cases, 2 groups of datasets need to be processed. With 3

cores, there are 2 cores stalling during the second group, which contains only one channel data being processed. Table 3.1(b) shows the computation speedup factors of the proposed method. Note that the proposed parallel computation provides about 1.3 times speedup when using 2 cores, and about 1.8 times speedup when using 4 cores, which are shown in Table 3.1(b) factor 128×128 data sets.

Table 3.1 (a) Total computation time (in seconds) in the simulated 4-channel study with a quad-core CPU when different numbers of CPU cores are used. (b) Computational speedup factors of the proposed method in the 4-channel study, as compared with the reconstruction using the quad-core CPU without the proposed method. Note that the acceleration is greater than one even with two cores using the proposed method [85].

(a)

time \ cores / size	1	2	3	4
16×16	12.3	7.7	7.4	6.2
32×32	17.8	13.2	11.3	10.1
64×64	40.3	26.8	26.3	22.5
128×128	125.0	76.8	81.2	59.3
256×256	567.0	325.6	334.6	248.9

(b)

speedup \ cores / data	1	2	3	4
128×128	0.8	1.4	1.4	1.8
256×256	0.8	1.3	1.3	2.0
512×512	0.7	1.2	1.1	1.6

Table 3.2 illustrates the total reconstruction time in the 4-channel simulated study in these parts: CS reconstruction (CS), computation starvation time (Stall), and communication time through file R/W (R/W). Comparing sub-plots for large image sizes with that of image size 32, the overheads of file R/W and the stall time take a larger percentage of the total computation time. However, when images become larger, the percentages of these overheads are dramatically reduced and the parallelization of CS reconstructions gains more benefits. Also, using 2 cores has less stall time, which is implicitly contained in CS reconstruction time, comparing to using 3 cores and 4 cores. Therefore, it provides maximum efficiency improvement per core, whereas using 3 cores has the minimum efficiency improvement per core.

Table 3.2(a) shows the time of image reconstructions of simulated 8-channel data and 16 channel data using a different number of cores of CPU. As shown in Table 3.2(b), the computation speedup factor of using the proposed method is calculated. The range of speedup factors in Table 3.2(b) is similar to that of Table 3.1(b), given minimum 0.8 times speedup and maximum 1.8 times speedup, compared with the conventional method using the quad-core CPU without the proposed parallelization.

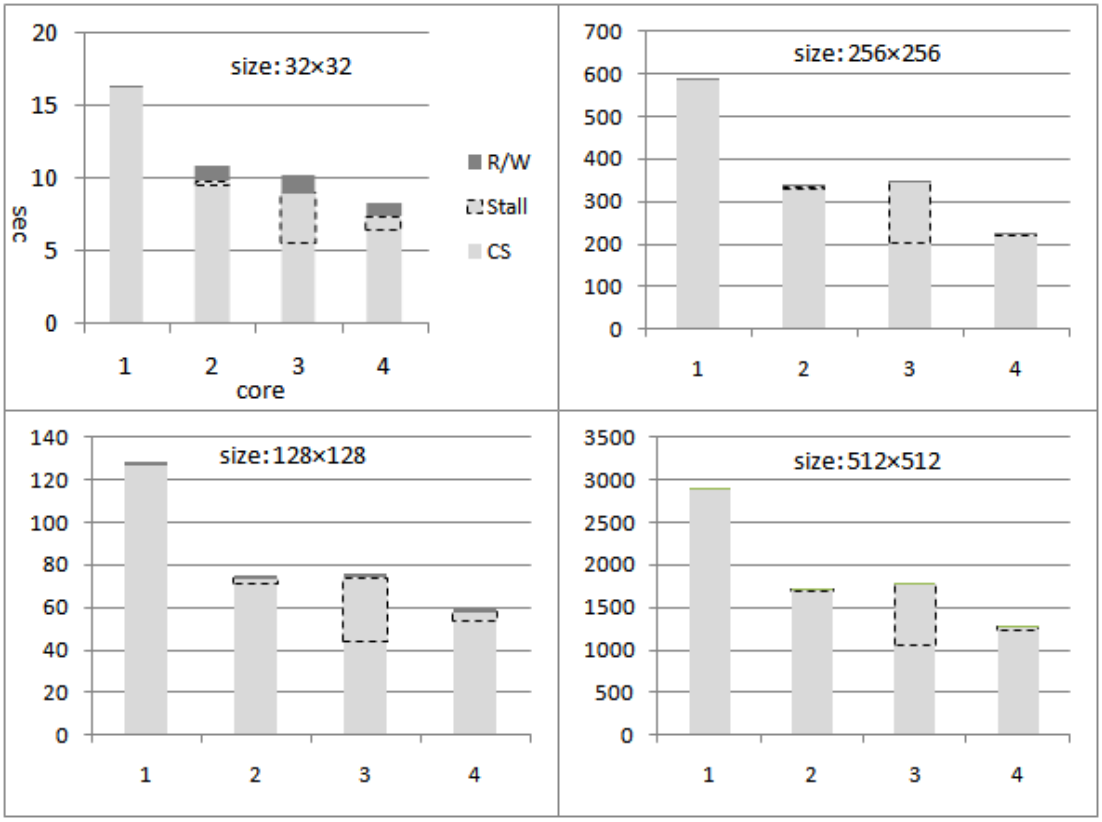


Figure 3.2 Total computation time shown in portions: CS reconstruction, computation starvation (cores stall), and file reading/writing. The horizontal axis represents the numbers of CPU cores in use and the vertical axis represents the computation time [85].

Table 3.2 Results of the simulated 8-channel and 16-channel studies: (a) total computation time (in seconds) when different numbers of cores are used; (b) Computational speedup factors, as compared with the reconstruction using the quad-core CPU without the proposed method [85].

(a)

time / cores / data	1	2	3	4
8-channel	1194.0	675.4	579.4	513.7
16-channel	2317.8	1358.1	1170.1	1017.5

(b)

speedup / cores / data	1	2	3	4
8-channel	0.8	1.3	1.6	1.7
16-channel	0.8	1.3	1.6	1.8

Figure 3.3 shows the image reconstruction in the 8-channel in-vivo human brain imaging experiments. The sum-of-squares reconstruction from the fully-sampled 256×256 data is shown in Figure 3.3(a), and the result from using the proposed method is shown in Figure 3.3(b). In this experiment, it took 718 seconds to reconstruct the image using the proposed algorithm, whereas it took 1161 seconds without the proposed algorithm. The speedup factor is about 1.6. Note that the image reconstruction quality used the proposed algorithm is closed to the one from the fully sampled data.

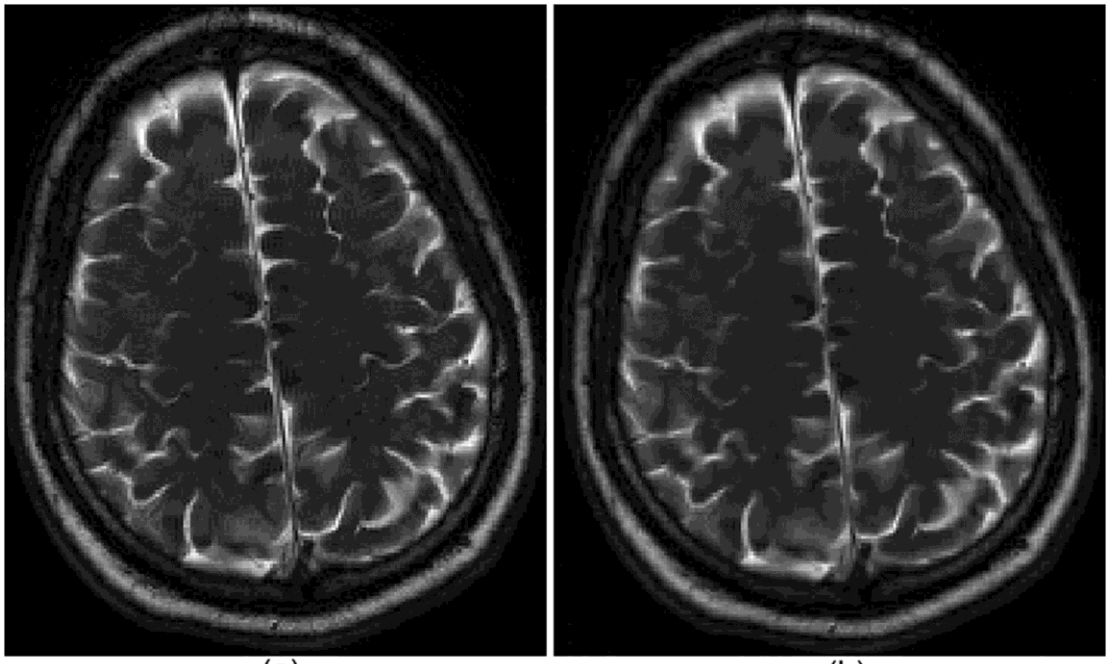


Figure 3.3 Images reconstructed from the 8-channel in-vivo data using (a) sum of squares from fully-sampled data, and (b) the proposed method from 33% of the total data. The time to reconstruct (b) is about 718 seconds, whereas it takes about 1161 seconds without the proposed method [85].

Discussion

We described an innovative reconstruction procedure for CS imaging with MR array receiver systems. The proposed method reconstructs images using the under-sampled data from each channel individually and combines them via the sum-of-squares method. Multi-core CPUs were used to reconstruct CS images simultaneously, which significantly shortened the reconstruction time. In our experiments with simulated 4-channel data, using 2 cores of CPU gave maximum efficiency improvement per core; while using 4 cores gave the fastest reconstruction. However, the efficiency was not doubled as we changed the

number cores from 2 to 4. This is because the memory bandwidth and the memory size are fixed and all cores share the same memory.

The proposed algorithm was tested for single slice 2D imaging in this work. However, it can be directly applied to multi-slice 2D imaging where parallelization can be used on multiple slices instead of multiple channels. For more computationally demanding cases such as 3D imaging or multi-channel multi-slice imaging, the efficiency gain from the parallelization will be even more beneficial. In CS image reconstruction studied in this paper, the overhead is only a small portion of the total reconstruction time. Significant computation time reductions were achieved using a quad-core CPU, especially for higher computation complexity of the reconstruction algorithms using wavelet transforms. Implementing the algorithms in C/C++ can further shorten the computational time significantly. Such improvement will be complementary to the gains achieved by parallelization. In addition, other existing methods using GPU architecture and more advanced synergetic integration of multi-core CPUs with GPUs can be potentially used to further accelerate the CS algorithms for 2D or 3D multi-channel data [76, 77]. These possibilities will be further explored in the future research to fully realize the potential of parallel computations for processing large, multi-channel data in MRI.

CHAPTER IV

IMPROVED COMPRESSED SENSING MRI WITH MULTICHANNEL DATA USING REWEIGHTED l_1 MINIMIZATION*

Objective

Imaging speed in magnetic resonance imaging (MRI) is an important issue, especially in the acquisition of diagnostic images in clinical settings because shortening the scan time can reduce the cost and increase throughput and patient's comfort. However, the data acquisition is practically limited by hardware capability and signal-to-noise (SNR) ratio factors. Compressed Sensing (CS) emerged as a method that allows a sparse signal to be reconstructed from a set of randomly undersampled projection data [67, 69]. It has been demonstrated that CS is useful for speeding up MRI acquisition, where data is collected in the k-space, i.e. Fourier space [32].

Multi-channel imaging using array receiver system offers improved SNR [7, 70] or accelerated speed with parallel imaging (PI). Therefore, integrating CS with PI are expected to further improve the MRI quality and/or speed [54-56, 71, 72]. In doing so, CS and PI coupled in a large linear system or decoupled in separate steps. In the latter case, CS algorithm is applied to each channel individually, then the final image can be reconstructed using the sensitivity encoding method [58] or a root-sum-of-square method [81]. In addition, the correlations of distributed compressed sensing have also been further

* Reprinted with permission from "Improving multi-channel compressed sensing MRI with reweighted l1 minimization" by Ching-Hua Chang and Jim Ji, 2014. *Quantitative Imaging in Medicine and Surgery*, vol. 4, pp. 19-23, Copyright 2014 AME Publishing Company.

applied in the system to improve the image quality [82]. In all the aforementioned methods, image reconstructions involve in minimizing the l_1 norm of a sparse image representation in certain domains, such as the wavelet domain or a sparse gradient. Since l_1 is an approximation of the sparsity measurements, i.e. l_0 norm of the sparse domain, there has been efforts to further improve l_1 minimization so that it will be closer to the l_0 minimization solution.

In this chapter, we develop a method that reconstructs MRI image from multi-channel data in the CS framework with a reweighted l_1 minimization. The main feature of the new method is that it uses an iterative, reweighted l_1 minimization method to perform the CS reconstruction of multi-channel MRI data. The method was compared with two existing multi-channel CS reconstruction methods using computer simulations and in-vivo MRI data. The results show that the proposed method can provide an improved reconstruction quality at a slightly increased computation cost. The content of this chapter is published on preliminary work presented in conference abstracts [81, 83], and on a journal paper[84].

Methods

The phased array MR receiver system consists of a set of receiver channels, which are individually connected to decoupled coil elements. With an array receiver system, a k-space data set $\boldsymbol{y}_k$, k = 1, 2,…, c, will be acquired from each channel. In applying CS reconstruction, each channel can be formulated as an underdetermined system, $\boldsymbol{y}_k = \boldsymbol{\Phi x}_k$, where $\boldsymbol{\Phi}$ is a randomly under-sample Fourier transform operator implemented by the

phase-encoding and frequency-encoding gradients. The CS theory states that an image $\boldsymbol{x}_k$ can be recovered from the incomplete k-space data $\boldsymbol{y}_k$ if it is sufficiently sparse. However, even the image itself is not sparse, it can often be transformed to a sparser domain and there is high probability that the image can be recovered. A commonly used sparsifying transform is the gradient operators; i.e. the image reconstruction can be achieved by solving the following convex optimization problem,

$$\min_{\mathbf{x}_k} TV(\mathbf{x}_k) \quad \text{s.t. } \mathbf{y}_k = \mathbf{\Phi}\mathbf{x}_k \qquad k = 1, 2, \ldots, \mathrm{c} \tag{4. 1}$$

where $TV(\mathbf{x}_k) = \sum_{\substack{1\le i\le \mathrm{m} \\ 1\le j\le \mathrm{n}}} \left\| (D\mathbf{x}_k)_{i,j} \right\|_2$, where $(D\mathbf{x}_k)_{i,j}$ represents the forward difference between adjacent pixels defined as $(\mathrm{x}_{i+1,j} - \mathrm{x}_{i,j}, \mathrm{x}_{i,j+1} - \mathrm{x}_{i,j})$. Here, total variation is considered as the l_1 norm of the magnitudes of the gradients and is well known as an isotropic version of total variation. This formulation follows the method described in [85]. After all channels images are reconstructed. They are combined using a root-sum-of-squares method. The overall reconstruction procedure is shown in Figure 4.1. As shown, under-sampled k-space data is fed to the use of l_1 minimization algorithm, whose outputs are recursively calculated as the weights of the next iteration and finally produce the final image.

In this paper, we utilize the reweighted l_1 minimization algorithm [86] to enhance the CS image reconstruction from multi-channel data. To solve the minimization problem in Eq. (4. 1), it is rewritten as a second-order cone problem with weights:

$$\min_{t_k, \mathbf{x}_k} \sum_{\substack{1 \le i \le m \\ 1 \le j \le n}} t_{k_{i,j}}$$

$$\text{s.t.} \quad w_{k_{ij}} \left\| (D\mathbf{x}_k)_{i,j} \right\|_2 \le t_{k_{i,j}} \qquad (4.\ 2)$$

$$\mathbf{y}_k = \mathbf{\Phi} \mathbf{x}_k$$

Where the weights are set to be inversely proportional to the signal magnitude. Based on the theory of reweighted l_1 minimization, the larger entries of $\boldsymbol{w}_k$, i.e., where signal magnitude is close to zero, will discourage small entries of the reconstructed image $\boldsymbol{x}_k$. In the proposed method, small weights are calculated from the previous reconstructed images. As a result, the weights can be considered as iterative parameters in the convex relaxation to improve the image reconstruction.

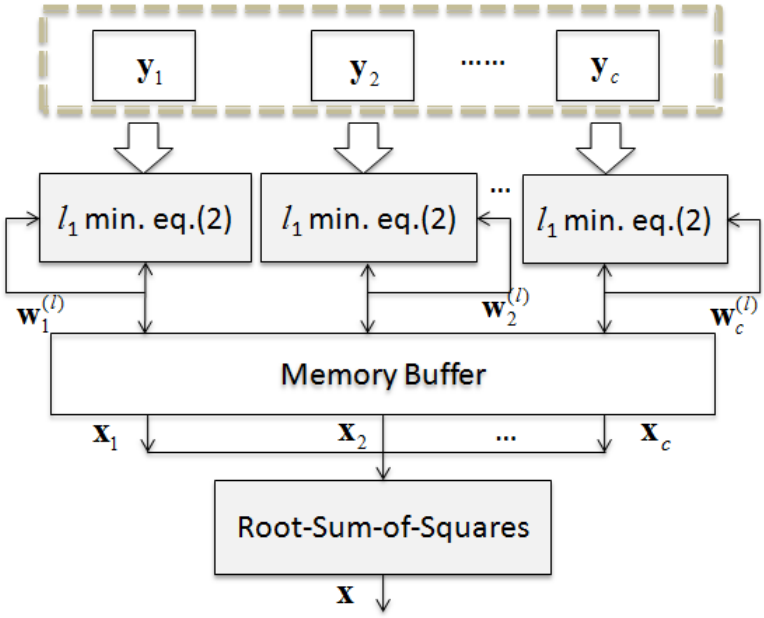


Figure 4.1 Reconstruction procedure for multi-channel receiver system using the l_1 reweighted minimization [84].

Specifically, each image $\boldsymbol{x}_k$ is reconstructed as follows.

1. Set the iteration count, $l = 1$ and the initial weight, $\mathrm{w}_{i,j}^{(1)} = 1$ for $i = 1, \ldots, \mathrm{m}$ and $j = 1, \cdots, \mathrm{n}$. Note that $\mathrm{w}_{i,j}^{(1)}$ is the weight on pixel (i, j).

2. Solve the weighted l_1 minimization problem

$$\mathbf{x}_k{}^{(l)} = \arg\min \sum_{\substack{1\le \mathrm{i}\le \mathrm{m} \\ 1\le \mathrm{j}\le \mathrm{n}}} \mathrm{w}_{k\,i,j}^{(l)} \left\|(\mathrm{D}\mathbf{x}_k)_{i,j}\right\|_2 \quad \text{s.t. } \mathbf{y}_k = \boldsymbol{\Phi}\mathbf{x}_k \tag{4. 3}$$

   This was performed using a home-made Matlab program by modifying the l_1-magic software package[46].

3. Update the weights:

$$\mathrm{w}_{i,j}^{(l+1)} = 1/\left(\left\|(\mathrm{D}\mathbf{x}_k^{(l)})_{i,j}\right\|_2 + \epsilon\right) \tag{4. 4}$$

   The parameter ε is a small positive number to prevent zero-valued denominator.

Finally, all the reconstruction images are combined by the root-sum-of-squares of all channel images.

$$\mathbf{x}(i,j) = \sqrt{\sum_{k=1}^{\mathrm{c}} |\mathbf{x}_k{}^{(l_{max})}(i,j)|^2} \tag{4. 5}$$

To test the proposed method, both simulated and in-vivo data were used. The k-space data of four channels were simulated using the 'Shepp-Logan' phantom with an image size of 128×128. The individual channel sensitivities are assumed to be shifted 2-dimension Gaussian functions. The individual channel data were under-sampled in the k-space with

radial sampling pattern. The under-sampling factor was about 15%, which meant only 15% of the total data were used in reconstructions. Finally, an 8-channel in-vivo brain MR data set was acquired and tested. The k-space data with an image size of 256×256 from each channel were acquired in full field-of-view, i.e., without under-sampling. Then, the radial sampling was simulated by decimation with an under-sampling factor of 25%. Based on the same sampling factor, the reconstruction image using the proposed method was compared with two methods: (1) conventional TV minimization (l_1 minimization with no reweighted iterations); and (2) a method shown in [71], which combine CS with SPACE-RIP (Sensitivity Profiles from an Array of Coils for Encoding and Reconstruction In Parallel).

The normalized means square error (NMSE) was used to evaluate the performance and defined as follows

$$\text{NMSE} = \left\|\hat{\mathbf{x}}_k - \mathbf{x}_k{}^{(l_{max})}\right\|_2 / \|\hat{\mathbf{x}}_k\|_2 \qquad (4.6)$$

Note that $\hat{\mathbf{x}}_k$ is the referenced image, which is reconstructed from the fully sampled data in the *k*-th channel.

Results

To show the quantitative improvement of the proposed approach the NMSE of the reconstructions by the conventional TV (l_1 minimization) and the proposed method is shown in Table 4.1. It shows that the proposed method has a lower NMSE than the conventional l_1 minimization algorithm since low NMSE represents fewer reconstruction

errors; the proposed method is superior in this study. Figure 4.2 to Figure 4.5 show the images and reconstruction details in the simulated phantom study.

Figure 4.2 indicates two regions and two lines of the original phantom study, which are used to compare the reconstructed details and resolutions. The comparisons of the reconstruction details are shown in Figure 4.3 and Figure 4.4. As the highlight region 1 and region 2 shown in Figure 4.3, the proposed method can recover more details of the edges pointed by the arrows. This is also illustrated in Figure 4.4, which displays the surface plots of the same corresponding zoom-in images shown in Figure 4.3. The three-dimension angle of view is also indicated along the arrows shown in Figure 4.3. Besides recovering sharper edges, it is observed that the proposed method can eliminate the staircase artifacts around smooth area noted by these arrows of Figure 4.4. In addition, the difference between the original image and the reconstructed image, i.e. reconstruction errors along line 1 and line 2, are shown in Figure 4.5. Again, it demonstrates the proposed method yields the reduced reconstruction errors.

Table 4.1 NMSEs of the image reconstruction in the simulated 4-channel phantom study [84].

NMSE	*Ch 1*	*Ch2*	*Ch3*	*Ch4*
TV (l_1 minimization)	0.015	0.041	0.036	0.016
Proposed (With reweighted l_1 minimization)	0.011	0.026	0.025	0.014

Table 4.2 NMSEs of the image reconstruction in the 8-channel in-vivo imaging experiment [84].

NMSE	*Ch2*	*Ch4*	*Ch6*	*Ch8*
TV (l_1 minimization)	0.092	0.086	0.096	0.086
Proposed (With reweighted l_1 minimization)	0.091	0.086	0.093	0.083

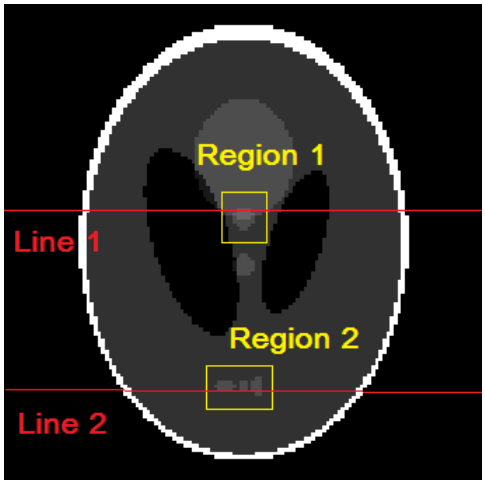


Figure 4.2 Original phantom image with selected regions and lines for comparisons [84].

Figure 4.6 compares the image reconstructions in an 8-channel in-vivo brain imaging experiment. Here, the left column represents the reconstructed images from the fully sampled data, the method in [71], where CS is integrated into a large linear system of multiple receiver coils, and the proposed method, respectively. The middle and right columns show the zoom-in views of the regions highlighted. To facilitate visualization, arrows are placed at the area where significant differences can be observed. As can be

seen, higher fidelity in details and sharper features are obtained with the proposed method. Note that all images in the middle and bottom rows are reconstructed from 25% of the fully sampled data.

A comparison between the proposed method and the conventional TV minimization is shown in Table 4.2 (Only even channels are shown). The performance in terms of NMSEs is shown. One can see that the proposed method has smaller quantitative reconstruction errors.

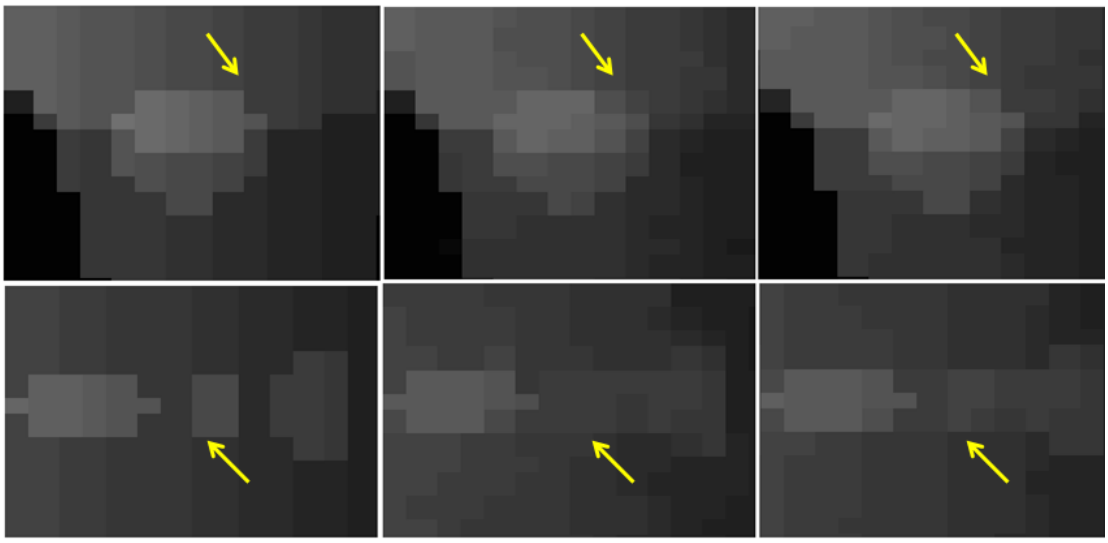


Figure 4.3 Reconstruction details of the Zoom-in region 1 (top row images) and region 2 (bottom row images) in the second channel image: (Left) Reference from the fully sampled data (Middle) with conventional l_1 minimization (TV) (Right) with the proposed reweighted l_1 minimization [84].

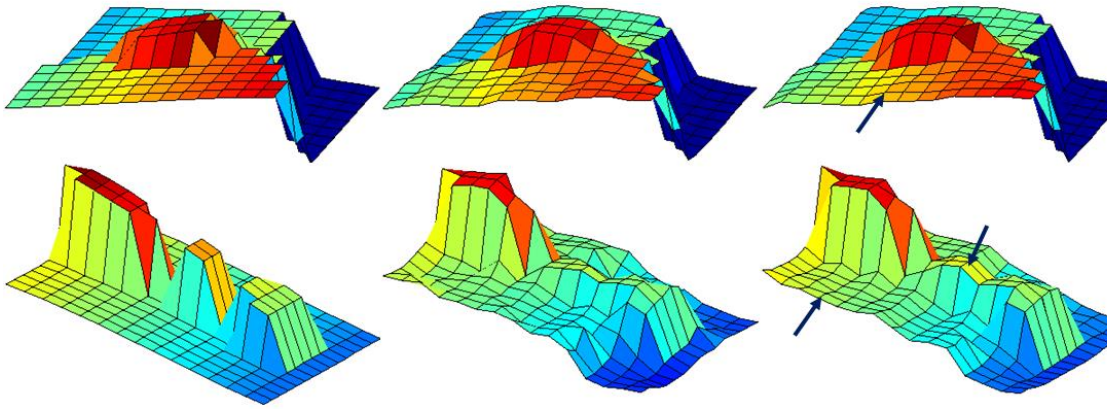


Figure 4.4 Surface plots of the corresponding zoom-in regions in Figure 4.3(Left) Reference from the fully sampled data (Middle) with conventional l_1 minimization (TV) (Right) with the proposed reweighted l_1 minimization [84].

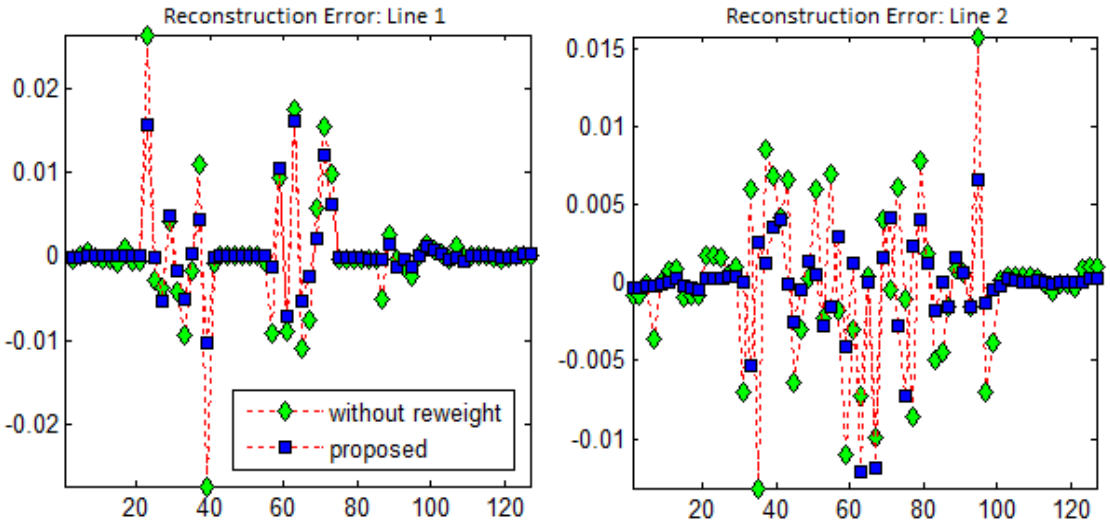


Figure 4.5 Reconstruction errors (differences between the original image and reconstructed image) along (Left) Line 1 (Right) Line 2 [84].

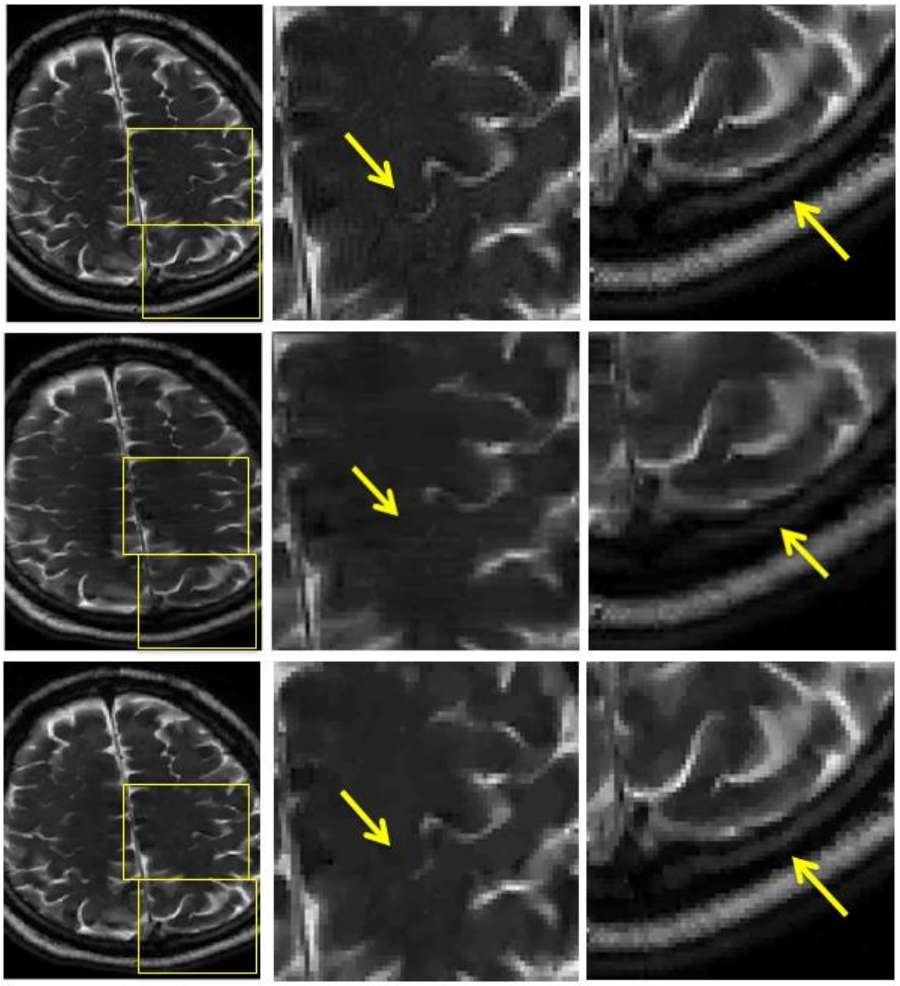


Figure 4.6 Images reconstructed from the 8-channel in-vivo data using (TOP) sum-of-squares from fully sampled data, (Middle) the method in [71], and (Bottom) the proposed method [84].

Discussion

A new improved reconstruction method for compressive sensing MRI with multi-channel phased array data was presented. In this method, the image is reconstructed using the reweighted l_1 minimization algorithm in a channel-by-channel fashion. The simulated experimental results show that the new method can provide an improved image quality

from the same data. On the other hand, the new algorithm requires more iterations than the conventional l_1 minimization algorithm. This might pose a problem when immediate delivery of images is preferred. In such cases, using multi-core processors such as graphics processing units (GPUs) can be applied to parallelize the reconstruction and to shorten the reconstruction time. The proposed method can also be applied to the other CS methods where l_1 minimization is used.

CHAPTER V

COMPRESSED SENSING RECONSTRUCTION FOR 3D MULTICHANNEL DATA USING GRAPHICS PROCESSING UNIT (GPU)

Objective

Conventionally, MRI acquires data in the spatial frequency domain, i.e. k-space, and perform the fast Fourier Transform to reconstruct the images. In this way, comparing to other imaging modalities, the scan time of fully sampling along Cartesian trajectories is usually unsatisfied. Clinically, longer scan time may cause patience's discomfort, and hence increase the imaging artifacts, such as motion blur.

Conceptually, reducing Nyquist rate can shorten the time of acquiring data. Non-Cartesian subsampling, such as radial sampling and spiral sampling, is one of this way, which directly uses interpolation or iterative method for image reconstruction[87, 88]. Another well-established technique for fast acquiring data is parallel imaging [17, 18, 20, 89]. It uses phased array receiver system to increase the frequency information and boots the image quality [7]. Compressed sensing (CS), on the other hand, provides a solid theory to reconstruct signals from fewer measurements, and therefore, it is of great interests in MRI applications [32, 69, 90]. The idea of combining CS with array receiver system has emerged because it can maximize the benefits of accelerating the data acquisition and/or improving reconstruction quality[54, 55, 71]. Liang et al. considered the combination as a two level reconstructions (CS-SENSE), and used CS reconstruction for random sampling followed by a SENSE reconstruction for uniform sampling [58]. In [65], a method of

autocalibrating parallel imaging (PI) was integrated into a nonlinear reconstruction of CS, well-known as l_1-SPIRiT. The implementation of 3D pseudorandom gradient echo sequence incorporating with l_1-SPIRiT has been proved that can highly speed up and/or improve image reconstructions compare to solely PI [64]. Otazo et al. presented a combination of k-t sparse with SENSE for first-pass cardiac perfusion [91]. She et al. proposed a novel iterative reconstruction which exploits the sparsity of both coil sensitivity and MR image in some sparse domains [62]. Those methods integrating CS into phased array receiver system require nonlinear regularization or l_1 minimization for image reconstructions. However, iterative reconstructions of CS MRI can be time-consuming, especially for 3D data and large array systems.

For the past few years, multicore processors have been used for accelerating image reconstructions in MRI, especially the applications and implementation in graphics processing units (GPUs). Borghi et al. solved and compared the l_1 minimization of CS reconstruction with different multicore platforms [75]. UIUC group has pioneered in the development of massively parallel computing on several advanced MRI reconstructions, such as speeding up a conjugate gradient for regularization problems, optimizing gridding algorithm and non-uniform Fourier transform for non-Cartesian sampling [92-94]. Sørensen et al. used GPU to accelerate gridding method followed by de-apodization for radial SENSE [95, 96]. Kim et al compared the implementation of SENSE-type regularization method on different platforms of multicore processors [97, 98]. Murphy et al. presented an implementation of l_1-SPIRiT, which requires dense lines in the center of k-space for auto-calibration signal (ACS), comparing the parallelization among multiple

CPUs and GPUs [66, 99]. In [100], based on gridding and de-apodization method, l_1 minimization for CS reconstruction of 3D radial sampled data is accelerated via GPU. Multicore CPU/GPU is used for accelerating the computations of patch-based directional wavelet or tight wavelet frame, which can improve the edge reconstruction in CS MRI [101, 102]. Comparing to the different image size of datasets, GPU speeds up the computations of CS MRI reconstructions using the split Bregman algorithm or the projection onto convex sets according to its requirements of GPU memory[103, 104]. GPU also speeds up the high-order singular decomposition method, which is used in the CS reconstructions of dynamic MRI [105]. In [106], GPU speeds up the computations of k-singular value decomposition and orthogonal matching pursuit, which are used for dictionary learning and training the sparse representations in CS MRI reconstructions. These previous work has shown powerful parallel computations of multicore processors and explored the speedup ability in different algorithms of reconstructing MRI images, which are computationally intensive with a large dataset. Most of them can achieve sub-minutes runtime for a wide variety of MRI reconstructions.

In our previous work [85], multi-core Central Processing Unit (CPU) was used to reconstruct multi-channel array data. Channel-by-channel CS reconstructions were pipelined in 4 cores and later combined by the sum-of-squares method. However, due to the limited number of cores, it still requires more than a minute to reconstruct a typical 2D image and only a moderate speedup factor (1.6-2.0) is possible. The runtime for higher dimensional multichannel MRI reconstruction is even longer. In this work, we used the same reconstruction flow. Instead of using nonlinear conjugate gradient method for l_1

minimization, an alternating direction algorithm is adopted, which is perfectly suitable for this reconstruction flow. All CS reconstructions can be completed in the GPU simultaneously. With proper data rearrangement in the memory, the element-wise operations are highly parallel processed by the streaming processors. Comparing to the above methods using GPU, our method requires no additional lines for ACS or the estimation of coil sensitivities from low-resolution images, whose accuracy usually affects the performance. Unlike gridding method, there are no approximation error in the iterative reconstructions. In addition, as the number of channel increases, it can provide an asymptotically optimal signal to noise ratio (SNR) because of the sum-of-square method [12]. The performance is compared with the runtime using CPU and the usage of GPU device memory is also analyzed in terms of the speedup ability. The content of this chapter is based on preliminary work presented in the conference abstract [107].

Methods

As CS is applied in MRI, fast scan time via subsampling is achieved by exploiting the sparsity of the signals. For large-scale data, the gain in shortening acquisition time may exchange with more reconstruction time without considering parallel computing. Taking three-dimensional (3D) k-space data with multiple channels, for instance, it practically requires up to several minutes for recovering 4D data with the conventional architecture of central processing unit (CPU). Using massively parallel processing, instead, the reconstruction time can be shortened in a clinical runtime.

The basic idea of the proposed parallel CS reconstruction is shown in Figure 5.1, which is based on a channel-by-channel CS reconstructions followed by a sum-of-square method (SOS). The best feature of the proposed method is that all channel images are decoupled and can be processed simultaneously. Especially, when the same acquisition method, e.g. Cartesian, a stack of spiral or a stack of radial, is applied on x-y slices for each z position, the reconstructions over each channel are regarded as multiple independent 2-D reconstructions.

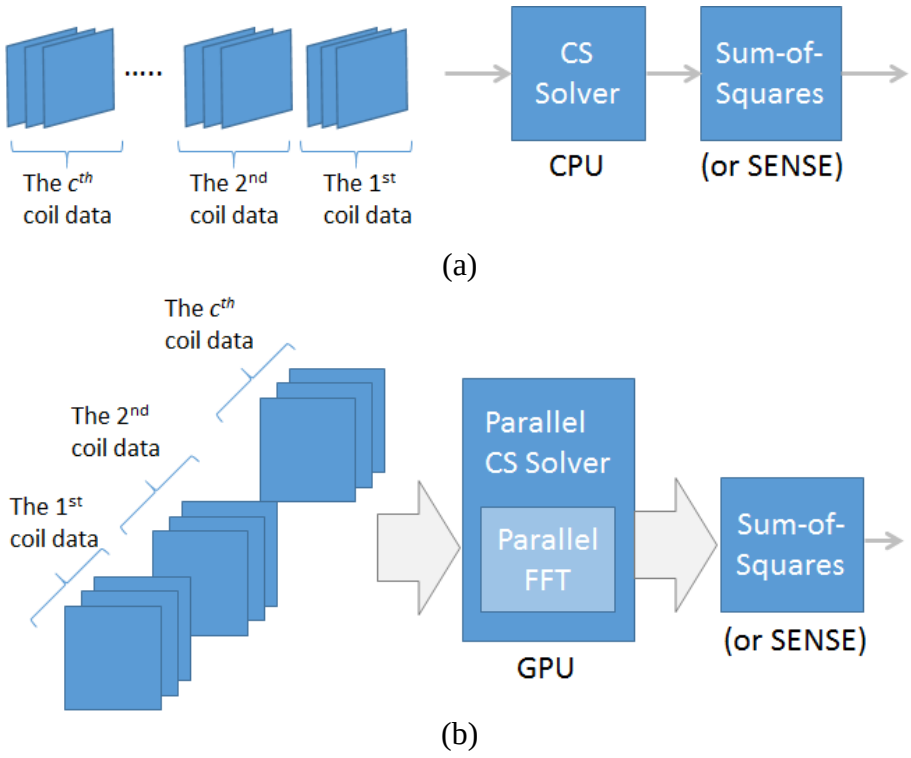


Figure 5.1 Work flow of (a) conventional sequential reconstruction and (b) parallel reconstruction with a GPU.

Figure 5.1(a) shows a workflow with the conventional reconstruction running on CPU without parallelization. Here, data from individual channels are processed sequentially and combined with SOS or Cartesian SENSE algorithm. Note that here the CS reconstruction is performed on individual channel data. Figure 5.1(b) shows the parallel CS reconstruction with GPU. As shown, datasets from all channels are transferred to the GPU memory before the execution of parallel CS solvers. This scheme allows massively parallel processing of all channels and slices.

Reconstruction Algorithm

In an MRI multiple receiver system, the k-space channel data, $\mathbf{v}_i \in R^M$, is received from a decoupled linear system, which can be formulated as

$$\mathbf{v}_i = SF\mathbf{u}_i = \Phi\mathbf{u}_i$$

Let Φ represent the combination of the subsampling operator S and the Fourier transform F, $\mathbf{v}_i$ be the randomly-acquired k-space data, $\mathbf{u}_i \in R^N$ be the unknown image of the i^{th} channel to be reconstructed, and c be the total channel number. According to CS theory, the channel image can be recovered by solving the following convex optimization problem:

$$\begin{aligned} &\min TV(\mathbf{u}_i) \\ &s.t.\ \|\Phi\mathbf{u}_i - \mathbf{v}_i\| < \epsilon \end{aligned} \tag{5. 1}$$

TV represents the total variation and is defined as $\sum_j \left\| D_j \mathbf{u}_i \right\|_2$, where the summation of the index j covers all pixels, ranging from 1 to N, and $D_j\mathbf{u}_i$ represents the vertical and horizontal finite differences of $\mathbf{u}_i$ at pixel j. Therefore, the objective function in eq.(5. 1) can be regarded as a sparsifying transform, and the minimization exploits the sparsity of

gradient coefficients among the i^{th} channel image. The second line in eq.(5. 1) defines a linear constraint for data consistency. The above linearly constrained problem can be formulated as minimizing a Lagrangian function. However, the *TV* term is not differentiable. Auxiliary variables, $\mathbf{p}_i = [(\mathbf{p}_i)_1 \quad \dots \quad (\mathbf{p}_i)_N]$ can be added to eq.(5. 1) to make the Eq.(5. 1) become an equality constraint.

$$\min_{\mathbf{u}_i, \mathbf{p}_i} \frac{\mu}{2} \|\Phi \mathbf{u}_i - \mathbf{v}_i\|_2^2 + \sum_j \left\|(\mathbf{p}_i)_j\right\|_2 \qquad (5.\ 2)$$
$$s.t.\ (\mathbf{p}_i)_j = D_j \mathbf{u}_i$$

Let $(\mathbf{p}_i)_j \triangleq D_j \mathbf{u}_i = \begin{bmatrix} D_j^{(1)} \mathbf{u}_i \\ D_j^{(2)} \mathbf{u}_i \end{bmatrix}$ is a 2-by-1 vector, and *D* is defined as $(D^{(1)}; D^{(2)})$, which is a 2*N*-by-*N* matrix, representing the horizontal and vertical finite difference operator at pixel *j* of the i^{th} channel. For example, assume that $\mathbf{u}_i$ represents the image with a size of 4×4, and its vector size is 16-by-1. As shown in Figure 5.2, the finite difference operator at pixel 6 (*j*=6), $D_6 = (D_6{}^{(1)}; D_6{}^{(2)})$ is a 2-by-*N* matrix.

For a 4×4 image: $\mathbf{u}_i = [1\ 2\ \dots 16]^T$

$$D_6^{(1)} = [0 \cdots 0 \quad 1_6 \quad 0 \quad 0\ 0 \quad -1_{10} \quad 0 \cdots 0]$$

$$D_6^{(2)} = [0 \cdots 0 \quad 1_6 \quad -1_7 \quad 0\ 0 \quad 0 \quad 0 \cdots 0]$$

Figure 5.2 Example of the finite difference operator for calculating the horizontal and vertical differences.

According to [108], the augmented Lagrangian function is applied to each i channel and the linear equality constraint is resolved by adding a penalty term to the objective function. Thus, eq. (5. 2) becomes an unconstrained problem,

$$\min_{\mathbf{u}_i,\mathbf{p}_i} \frac{\mu}{2}\|\Phi\mathbf{u}_i - \mathbf{v}_i\|_2^2 + \sum_j \left\|(\mathbf{p}_i)_j\right\|_2 + \boldsymbol{\lambda}_i{}^T(\mathbf{p}_i - D\mathbf{u}_i) + \frac{\beta}{2}\|\mathbf{p}_i - D\mathbf{u}_i\|_2^2 \tag{5. 3}$$

Here, the variable $\boldsymbol{\lambda}_i$ is an estimate of the Lagrange multiplier. Instead of solving the augmented Lagrangian function, this l_1-$l_2$convex function can be regarded as minimizing two sub-problems based on the alternating direction method of multipliers (ADMM): l_1 sub-problem with respect to $\mathbf{p}_i$ and l_2 sub-problem with respect to $\mathbf{u}_i$. For l_1 sub-problem, $\mathbf{u}_i$ and $\boldsymbol{\lambda}_i$ are fixed, and the iterative shrinkage/thresholding method is performed for solving sparse coefficients. i.e. finite difference when total variation is used as a sparsifying transform,

$$(\mathbf{p}_i)_j = \text{shrink}(D_j\mathbf{u}_i + \frac{(\boldsymbol{\lambda}_i)_j}{\beta}, \frac{1}{\beta}) \tag{5. 4}$$

which is known as 2D shrinkage function according to the definition of *TV*.

$$\text{shrink}\left(\mathbf{w}, \frac{1}{\beta}\right) = \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \cdot \max(\|\mathbf{w}\|_2 - \frac{1}{\beta}, 0) \tag{5. 5}$$

For l_2 sub-problem, $\mathbf{p}_i$ and $\boldsymbol{\lambda}_i$ are fixed, and the original eq.(5. 3) becomes a least-squares problem.

$$\min_{\mathbf{u}_i} \frac{\mu}{2} \|\Phi \mathbf{u}_i - \mathbf{v}_i\|_2^2 + \boldsymbol{\lambda}_i{}^T (\mathbf{p}_i - D\mathbf{u}_i) + \frac{\beta}{2} \|\mathbf{p}_i - D\mathbf{u}_i\|_2^2 \tag{5. 6}$$

By taking the derivative with respect to $\mathbf{u}_i$, finding the minimum of eq.(5. 6) is equivalent to solving the following normal equation

$$\mu \Phi^T (\Phi \mathbf{u}_i - \mathbf{v}_i) + (\boldsymbol{\lambda}_i{}^T D)^T + \beta D^T (\mathbf{p}_i - D\mathbf{u}_i) = 0$$

$$\left(D^T D + \frac{\mu}{\beta} \Phi^T \Phi\right) \mathbf{u}_i = D^T \left(\mathbf{p}_i - \frac{\boldsymbol{\lambda}_i}{\beta}\right) + \frac{\mu}{\beta} \Phi^T \mathbf{v}_i \tag{5. 7}$$

To solve eq.(5. 7) more efficiently, $D^{\mathrm{T}} D$ is further diagonalized by the 2D discrete Fourier Transform, F.

$$F\left(D^T D + \frac{\mu}{\beta} \Phi^T \Phi\right) F^T F \mathbf{u}_i = F D^T \left(\mathbf{p}_i - \frac{\boldsymbol{\lambda}_i}{\beta}\right) + \frac{\mu}{\beta} F \Phi^T \mathbf{v}_i$$

$$\left(F D^{\mathrm{T}} D F^{\mathrm{T}} + \frac{\mu}{\beta} \mathrm{S}^T \mathrm{S}\right) F \mathbf{u}_i = F D^T \mathbf{p}_i + \frac{\mu}{\beta} \mathrm{S}^T \mathbf{v}_i \tag{5. 8}$$

Where

$$F D^T D F^T = F[D^{(1)} \quad D^{(2)}] \begin{bmatrix} D^{(1)} \\ D^{(2)} \end{bmatrix} F^T \tag{5. 9}$$

$$= F D^{(1)^T} D^{(1)} F^T + F D^{(2)^T} D^{(2)} F^T$$

$$= F D^{(1)^T} (F D^{(1)^T})^T + F D^{(2)^T} (F D^{(2)^T})^T$$

$F D^{(1)^T}$ and $F D^{(2)^T}$ represent the vertical and horizontal finite difference operator in frequency domain, so that the left-hand side of eq. (5. 8) becomes element-wise operation for $F\mathbf{u}_i$. As a result, the iterative algorithm of solving the minimization problem of eq.(5. 1) for each *i* channel is shown as follows:

1. Setting the iteration count, $k = 0$, and the initial multipliers, $\mu > 0, \beta > 0, \gamma \in (0, \frac{\sqrt{5}+1}{2})$, $\boldsymbol{\lambda}_i = \boldsymbol{\lambda}_i{}^0, \mathbf{u}_i = \mathbf{u}_i{}^0$.
2. Solving the l_1 subproblem by shrinkage method: $(\mathbf{p}_i)_j{}^{k+1} = \text{shrink}(D_j \mathbf{u}_i{}^k + \frac{(\lambda_i)_j{}^k}{\beta}, \frac{1}{\beta})$
3. Solving the l_2 subproblem by solving eq. (5. 8): calculating $\mathbf{u}_i{}^{k+1}$, where $(\mathbf{p}_i, \boldsymbol{\lambda}_i) = (\mathbf{p}_i{}^{k+1}, \boldsymbol{\lambda}_i{}^k)$.
4. Update multipliers of ADM, $\boldsymbol{\lambda}_i{}^{k+1} = \boldsymbol{\lambda}_i{}^k - \gamma\beta(\mathbf{p}_i{}^{k+1} - D\mathbf{u}_i{}^{k+1})$
5. If $k < k_{\max}$ and $\frac{\|\mathbf{u}^{k+1} - \mathbf{u}^k\|_2}{\|\mathbf{u}^k\|_2} <$ tolerance, increase *k* and go to step 2.

Finally, all the reconstructed images, $\mathbf{u}_i$, are combined by the SOS method, which computes the root-mean-square average of the channel images.

$$\mathbf{u} = \sqrt{\sum_i |\mathbf{u}_i{}^{(k_{max})}|^2} \qquad (5.\ 10)$$

This iterative algorithm converges rapidly and performs few operations for each iteration.

GPU Implementation

The above algorithm is implemented on a graphics processing unit (GPU) platform. Since power dissipation of cooling system limits the development of sequential microprocessors, further acceleration of computations via increasing clock frequency was impractical. The current trend of developing microprocessors is toward multi-core and many core models. Although the move towards massively parallel computing with multi-core processors

increases the computational power, it also poses a challenge of speeding up applications to software developers. Particularly, an efficient implementation of CS MRI reconstructions with the multicore systems, such as GPU, requires the specific design of parallelization and optimization.

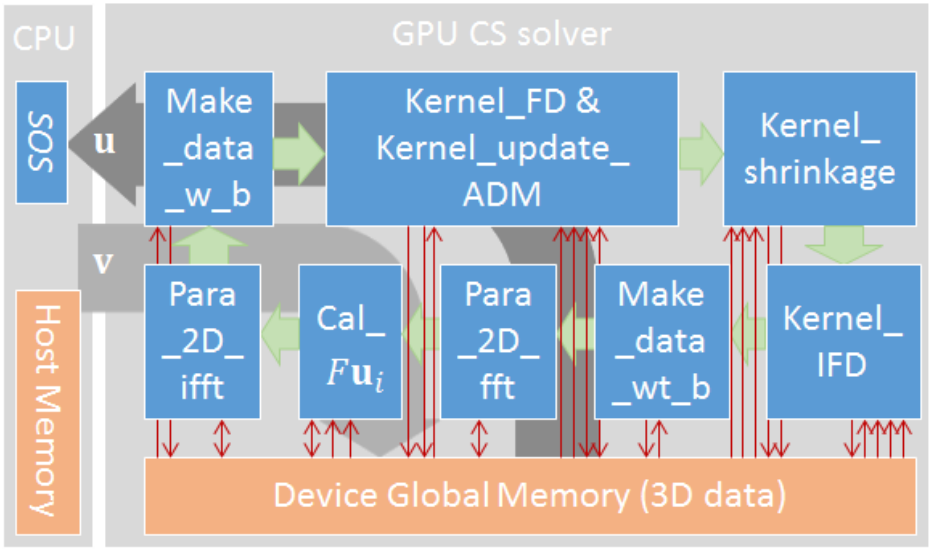


Figure 5.3 Data flow diagram of the parallel reconstructions using GPU. Gray thick arrows indicate data transferring between devices, gray thin arrows indicate save/load from global memory, and green arrows indicate data flow direction of iterations.

Figure 5.3 shows the parallel reconstruction method designed with GPU. The blue blocks illustrate the kernel functions (SOS is implemented on CPU). There are eight kernel functions programmed on the GPU. There are eight kernel functions programmed on GPU. For $l_1$sub-problem, the kernel functions include

1) Make_data_w_b: making data with boundary and separating the real and imaginary parts of the complex dataset.

2) Kernel_FD & Kernel_update_ADMM: calculating finite difference and updating ADM multipliers.
3) Kernel_shrinkage: element-wise operator for calculating sparse coefficients.
4) Kernel_IFD: calculating inverse finite difference.

For $l_1$sub-problem, the kernel functions include

1) Make_data_wt_b: making data without boundary and rearranging the complex data to be interleaved.
2) Para_2D_fft: launching parallel 2D FFT to calculate the first term in the right-hand side of Eq.(5. 8).
3) Cal_ $F\mathbf{u}_i$: calculating $F\mathbf{u}_i$ in the left-hand side of Eq.(5. 8).
4) Para_2D_ifft: launching parallel 2D IFFT to calculate the channel image $\mathbf{u}_i$.

Because finite difference matrices are diagonalized by discrete Fourier transform, they are actually circulant operators according to eq.(5. 9). As a result, one-pixel periodic boundary conditions are required for data, $\mathbf{u}_i$, so that the stencil computation can be applied to each pixel for calculating finite difference (FD) and inverse finite difference (IFD). There are two data types defined in the device global memory. One is float/double for the data flow in Kernel_FD & Kernel_update_ADMM, Kernel_shringkage and Kernel_IFD, where the real and imaginary parts of complex datasets are declared separately in the device memory; the other one is cufftComplex/ cufftDoubleComplex for the data flow in Para_2D_fft, Cal_$F\mathbf{u}_i$ and Para_2D_ifft, where the complex datasets are declared in an interleaved form in the device memory. Therefore, Make_data_w_b is designed for making data, $\mathbf{u}_i$, with

boundary and separating the real and imaginary parts of $\mathbf{u}_i$. The advantage is that real and imaginary parts of the datasets share same instructions as calculating FD and IFD, so that the number of threads are doubled. On the other hand, in order to launch multiple parallel FFTs and IFFTs by utilizing CUDA FFT library, it is necessary to prepare complex data in an interleave format. Therefore, Make_data_wt_b is designed for making data without boundary and rearranging the complex data to be interleaved.

Red arrows indicate the data access of kernel functions, each representing one variable of 3D dataset saved or loaded from the device global memory. The green thick arrows indicate the data flow direction of iterative CS reconstruction. All acquired data, $\mathbf{v}$, are transferred from CPU memory to GPU global memory at the beginning, indicated by a thick gray arrow. After all 3D channel images, $\mathbf{u}$, are iteratively reconstructed by GPU CS solver, they are transferred from the GPU global memory to the CPU memory. This prevents frequent data access between devices from slowing down GPU. Note that all acquired data, $\mathbf{v}_i$, are transferred from CPU memory to GPU global memory at first. Till GPU CS solver iteratively reconstructs all the channel images, $\mathbf{u}_i$, they can be transferred once from GPU global memory to CPU memory at last. That is because frequently access data between device and host memory will stop GPU from responding and trigger the OS to recover the device. The gray thin arrows between GPU memory and kernel functions represent the number of read and write for calculating the output of each pixel. 3D multichannel MRI datasets are randomly undersampled according to the central-weighted sampling along two phase encoding directions, and 1D IFFT can be applied along frequency encoding direction first. Therefore, multiple channels and slices of a high

dimensional MRI data are highly parallelized, and operations of CS reconstructions can be simultaneously processed by SMX.

In addition, Kernel_FD & Kernel_update_ADMM are actually combined together. The flow chart of GPU kernel codes for Kernel_FD & Kernel_update_ADMM are shown in Figure 5.4 and Figure 5.5. While Figure 5.4 shows the flowchart of kernel function without using shared memory, Figure 5.5(a) presents the kernel function using shared memory. The GPU kernel codes of Figure 5.4 deals with changing thread indices to a form of column-row indices. Then, if indicating correct memory address, the thread can compute FD and update ADM multipliers. On the other hand, a common strategy, known as tiling, is required to partition the data into tile blocks when using the shared memory. The concept of tiling is illustrated in Figure 5.5(b). Data **u** is originally partitioned according to the size of thread blocks. Normally, the size of tiles indicated as orange blocks in Figure 5.5(b) is set according to the size of thread block, and each tile fits into the shared memory. In this paper, BLOCK_SIZE represents the size of thread block. If the BLOCK_SIZE is set to 4, the TILE_WIDTH will be 5, which is equal to BLOCK_SIZE+1, due to the requirement for the value of right and down pixels in calculating FD. The size of array $s\mathbf{u}$ declared in shared memory is set to match the tile size. After the data of tile blocks are copied from global memory to the shared memory, threads indicated as blue arrows can compute FD and updates the ADM multipliers. The settings for using tile blocks and the shared memory are emphasized with gray color in Figure 5.5(a). Because almost all elements of **u** are accessed three times, this strategy can reduce the traffic of accessing GPU global memory nearly by one-third. The function call, __syncthreads(), is

a barrier to avoid race condition and make sure that all data is loaded into the shared memory ready for accessing. Threads in the same block will wait at the calling location until each one reaches where the barrier locates.

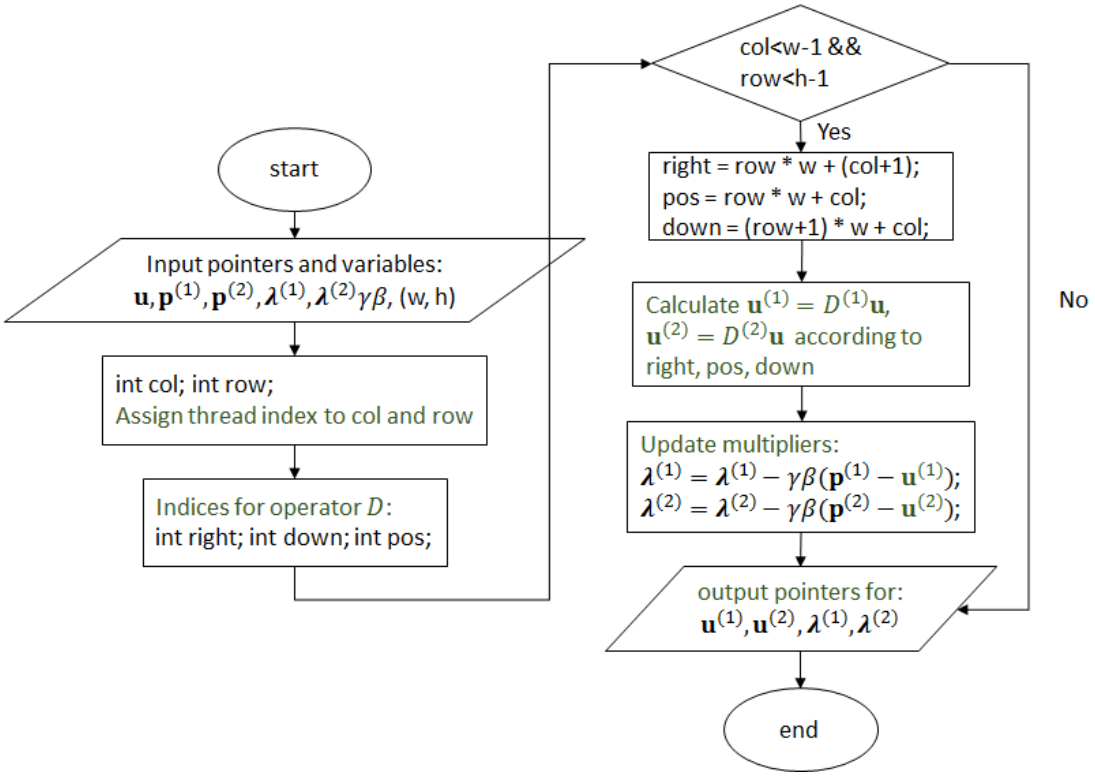
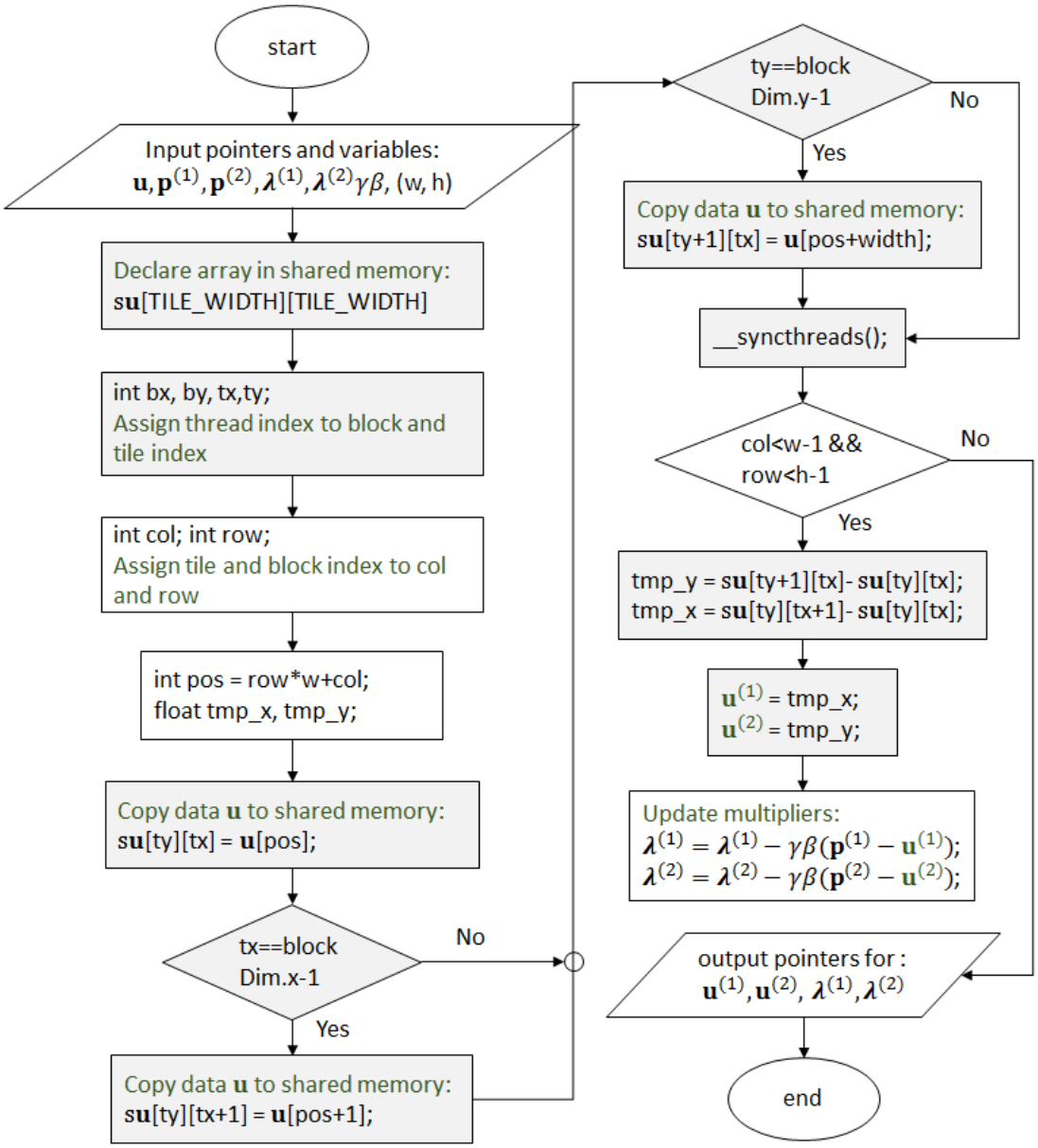
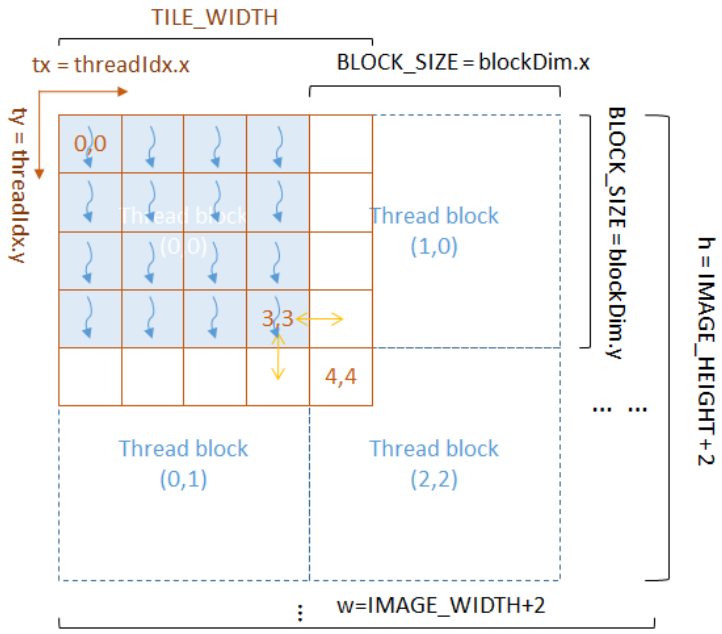


Figure 5.4 Flowchart of Kernel_FD & Kernel_updata_ADMM without using shared memory.



(a)

Figure 5.5 (a) Flowchart of Kernel_FD & Kernel_updata_ADMM using shared memory. (b) Illustration of thread blocks and the tile block.



(b)

Figure 5.5 Continued.

Another kernel function, Kernel_IFD, can also take advantage of using device shared memory. There are five methods shown in Figure 5.6 to Figure 5.10. Figure 5.6 presents the GPU codes without using shared memory. In the function of Kernel_IFD, there are four arrays, $\mathbf{p}^{(1)}$, $\boldsymbol{\lambda}^{(1)}$, $\mathbf{p}^{(2)}$, $\boldsymbol{\lambda}^{(2)}$ representing the horizontal and vertical FD(sparse coefficients) and the corresponding multipliers. The thread indices are changed to a form of column-row indexing. Then, if the addresses classified by 'col' and 'row' are within the data boundary, threads compute the value of $D^{\mathrm{T}}\left(\mathbf{p}_i - \frac{\lambda_i}{\beta}\right)$ in eq. (5. 8) if the addresses indicated by 'col' and 'row' are within the data boundary. Note that The parameter β has been preprocessed on $\mathbf{p}_i$ or $\boldsymbol{\lambda}_i$, so that the division can be replaced by the multiplication and allocated after the operation of FD. In Figure 5.6 using the shared

memory, four arrays, $s\mathbf{p}^{(1)}$, $s\boldsymbol{\lambda}^{(1)}$, $s\mathbf{p}^{(2)}$, and $s\boldsymbol{\lambda}^{(2)}$ refer to $\mathbf{p}$ and $\boldsymbol{\lambda}$ in shared memory. Because each elements of vertical and horizontal arrays is read twice, the total number of accesses to the global memory can be reduced nearly by half. One concern is that the size of array in use could exceed the capacity of the shared memory with the memory size limitation. This could lead to the reduction of the maximum number of thread blocks being processed in each SMX and therefore deteriorate the performance. It can be improved by narrowing down the sizes of thread and tile blocks or using shared memory more efficiently.

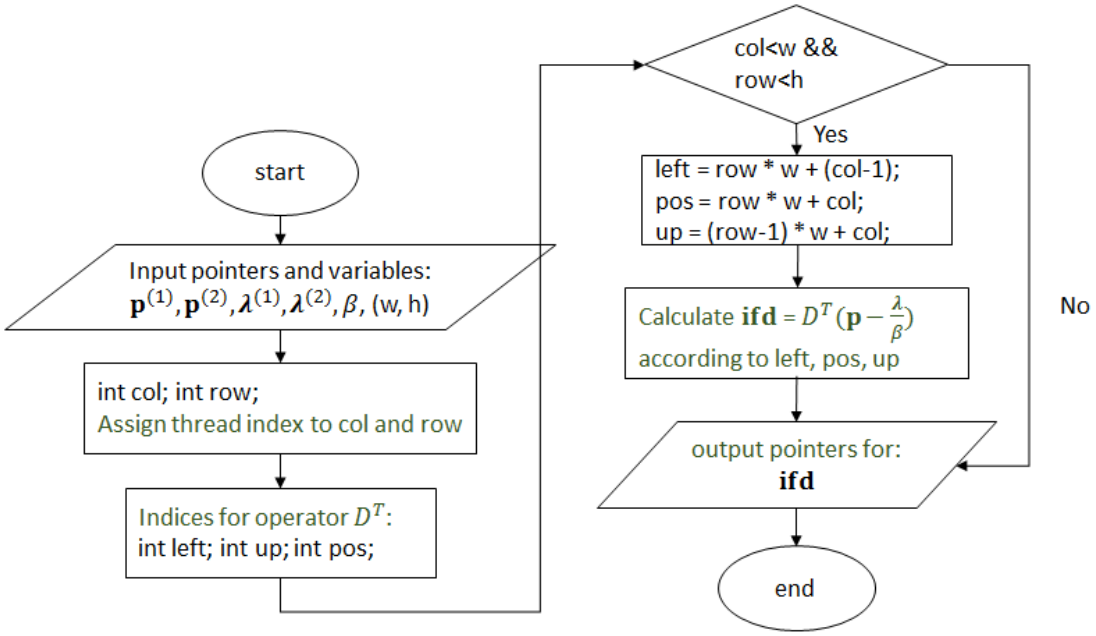
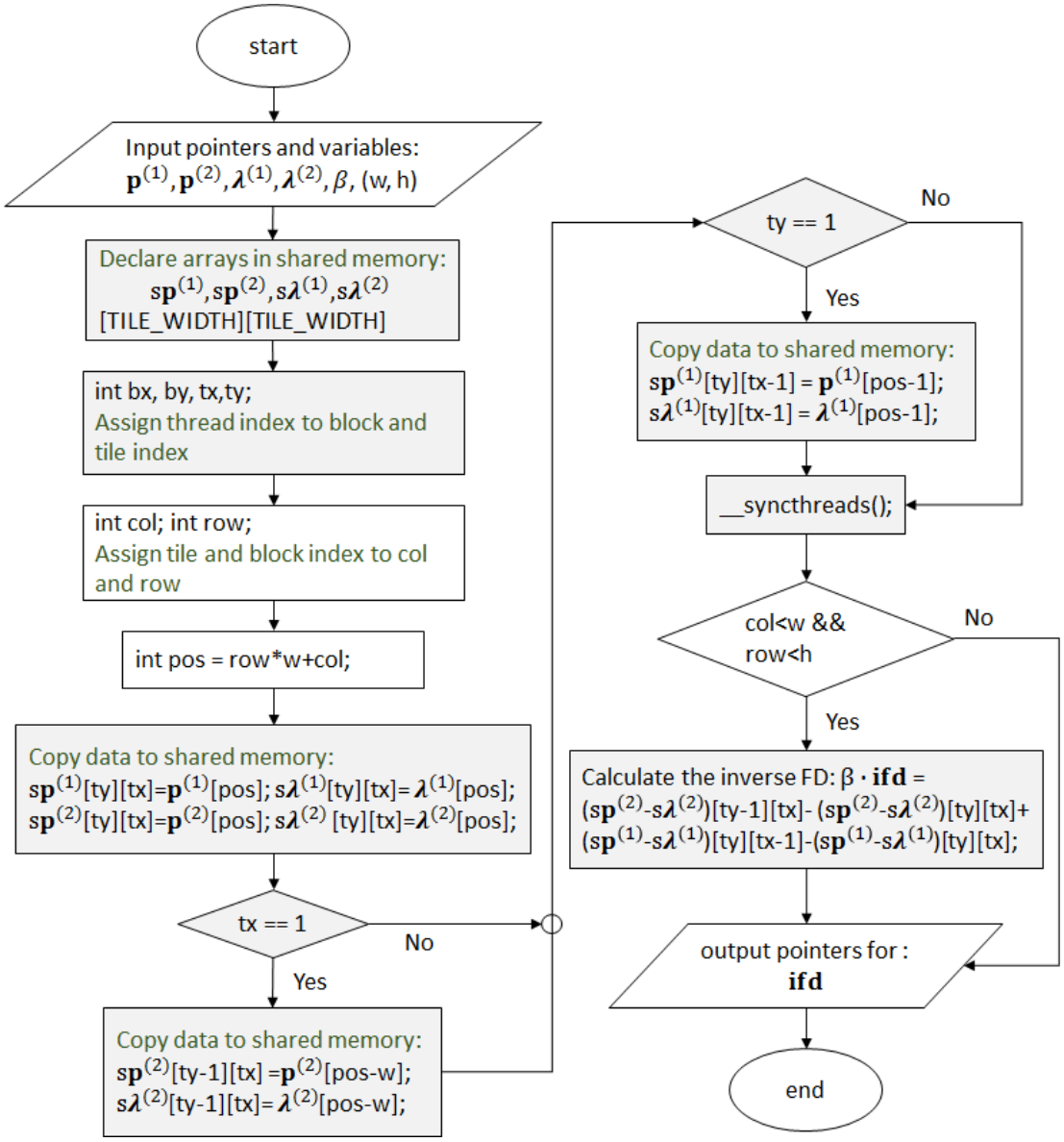
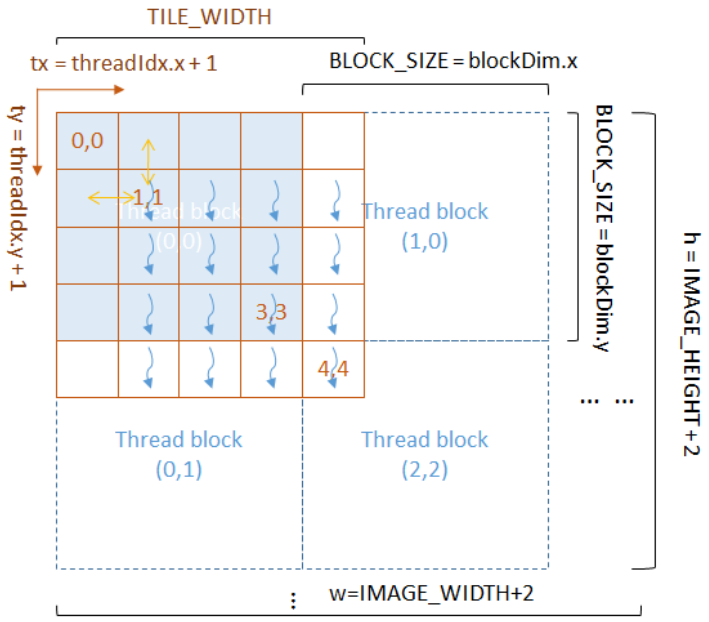


Figure 5.6 Flowchart of GPU codes for Kernel_IFD without using shared memory.



(a)

Figure 5.7 (a) Method 1 – Flowchart for Kernel_IFD using 4 arrays in shared memory. (b) Illustration of tiling concepts and the corresponding data.



(b)

Figure 5.7 Continued.

The illustration in Figure 5.7(b) expresses the mappings of the tile block and thread block to the global memory colored in orange and blue respectively. Threads of execution are shifted to the right and down by setting tx=threadIdx.x+1 and ty=threadIdx.y+1. Therefore, two more passes should be added for assigning data to the boundary of $s\mathbf{p}^{(2)}$, $s\boldsymbol{\lambda}^{(2)}$, $s\mathbf{p}^{(1)}$,or $s\boldsymbol{\lambda}^{(1)}$ when 'tx' or 'ty' is equal to 1. Because the index, 'pos', is added by one column and row, the corresponding position in global memory is shifted to the right and down by one. Thus, it should avoid to accessing the memory locations, which will potentially exceed the physical range of the global memory. 'Col' and 'row' represent the indices of shifted thread block, setting the conditions of 'col<w' and 'row<h'.

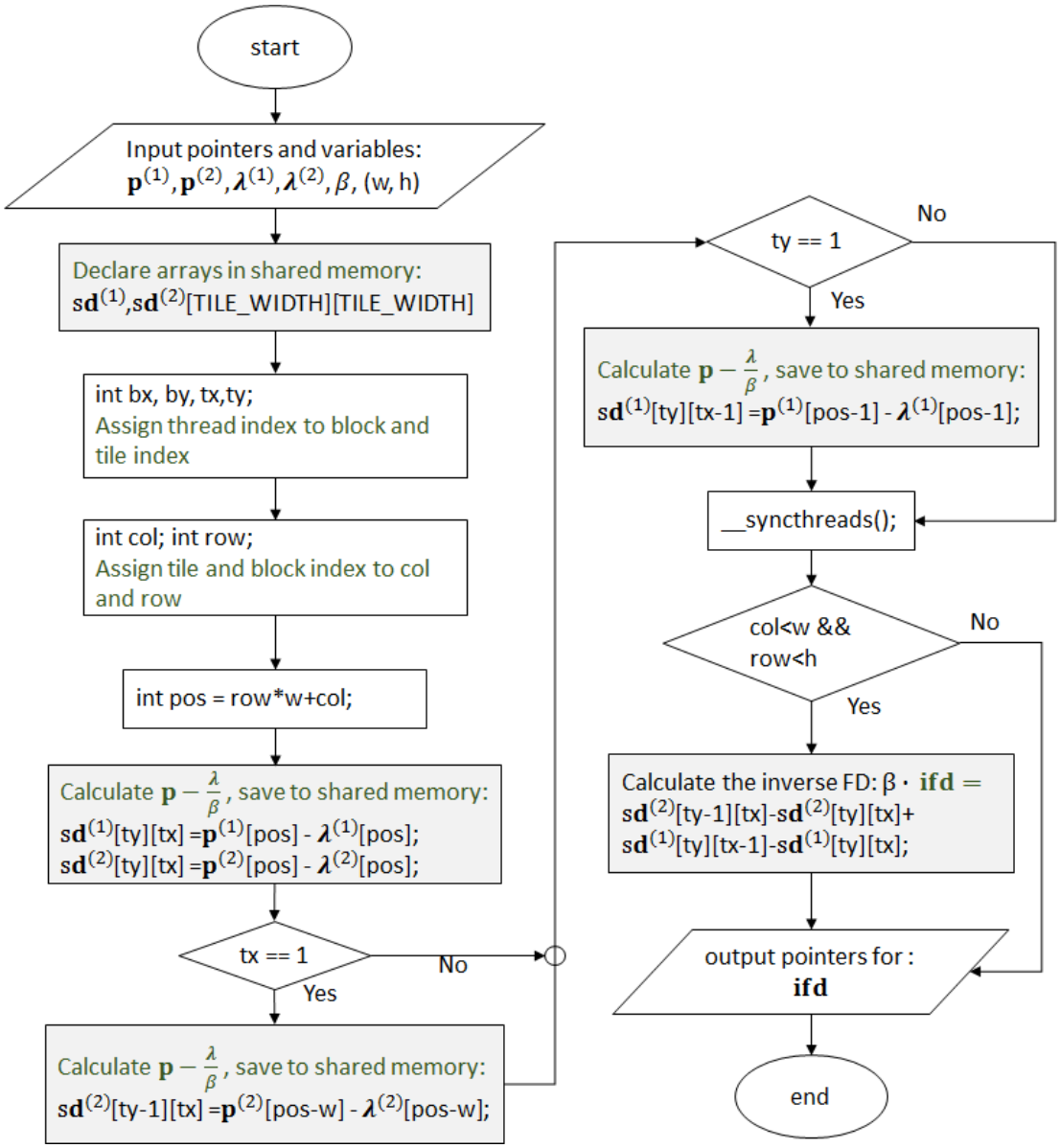
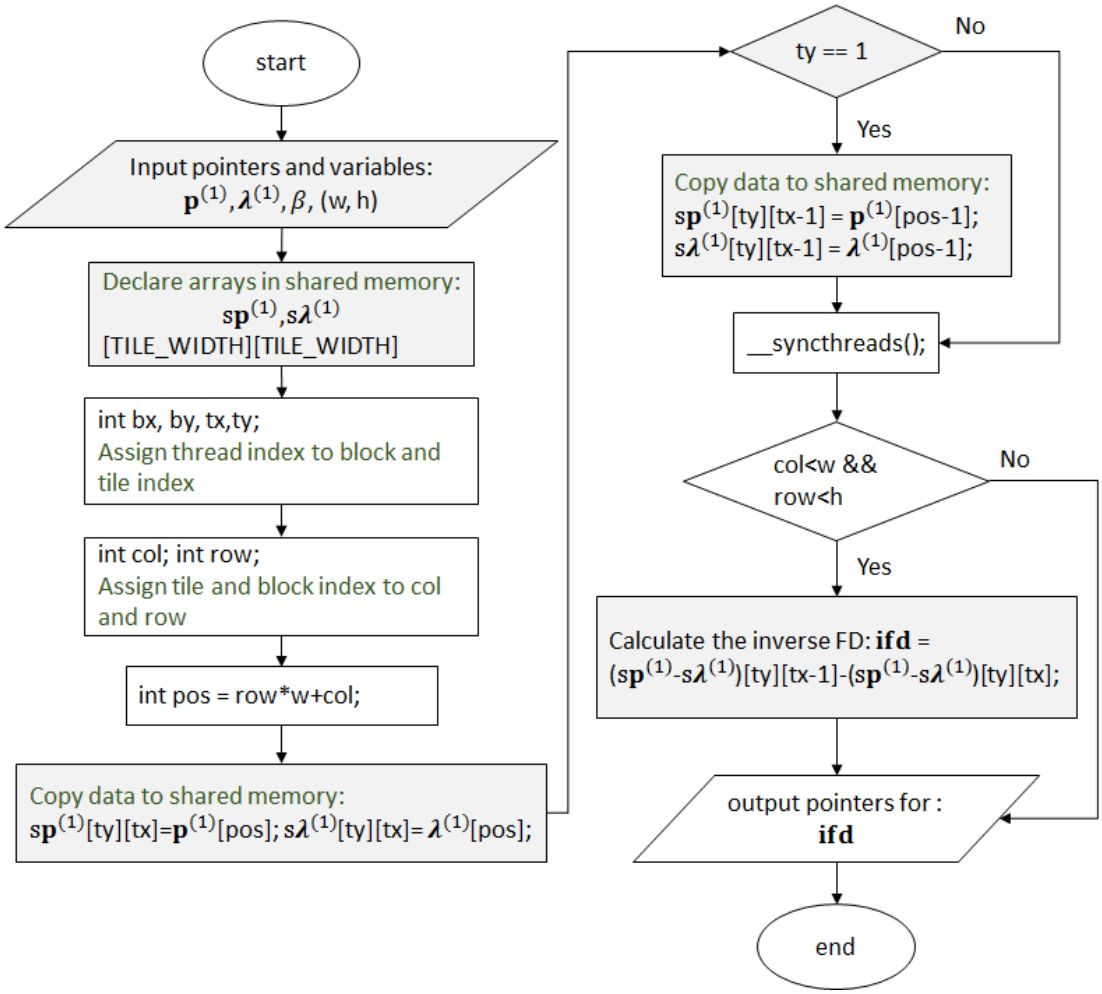


Figure 5.8 Method 2 – Flowchart for Kernel_IFD using 2 arrays in shared memory.

Instead of declaring 4 arrays in the shared memory, method 2 shown in Figure 5.8 calculates $\left(\mathbf{p}_i - \frac{\lambda_i}{\beta}\right)$ initially, and saves the horizontal and vertical difference to two arrays in the shared memory, $s\mathbf{d}^{(1)}$ and $s\mathbf{d}^{(2)}$, so that the numbers of arrays declared in the shared

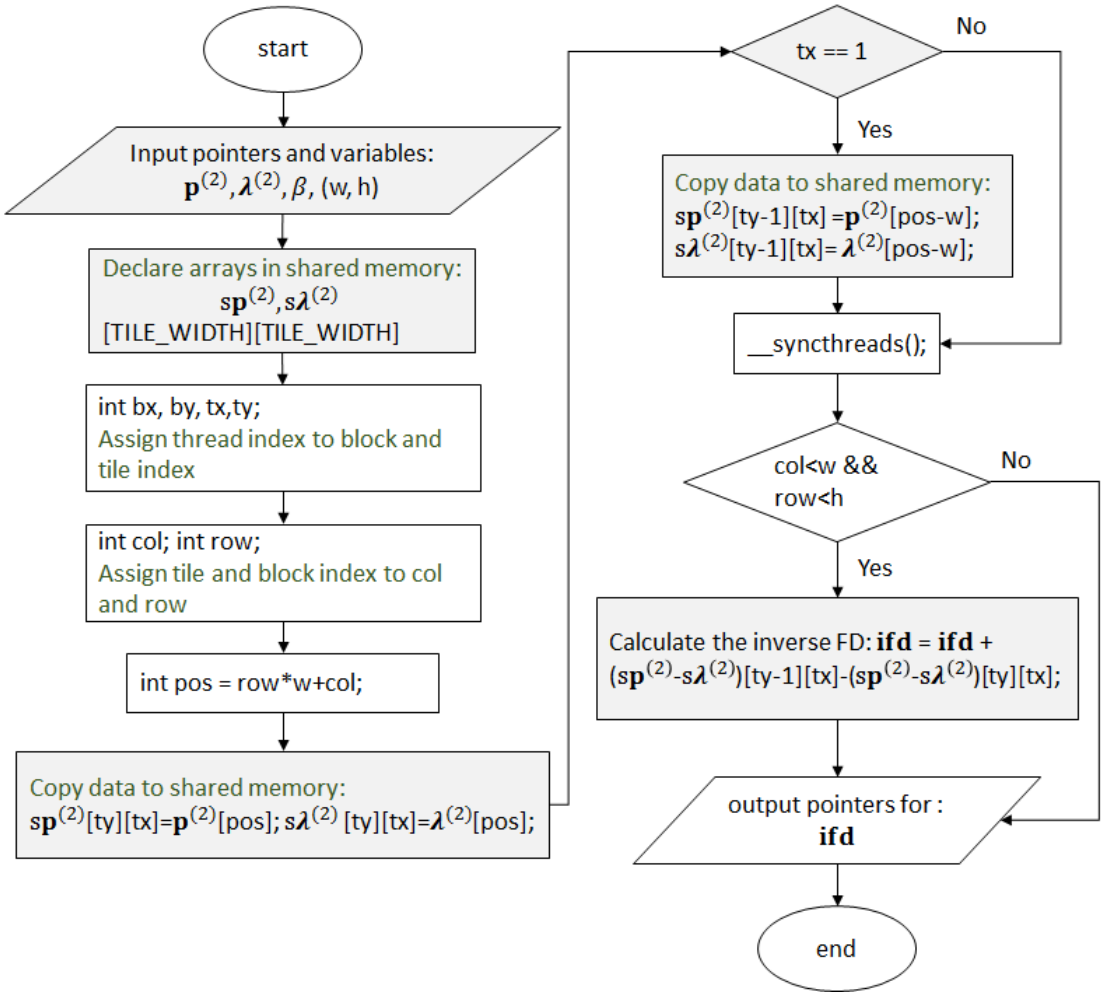
memory can be reduced from 4 to 2. The method without using shared memory, using shared memory with Method 1, and with Method 2 are compared to see how the number of block arrays declared in shared memory effects the performance. Besides, the effect of reducing bandwidth for loading data from device global memory will be revealed according to the comparisons among these methods.

Figure 5.9 shows the Method 3 using 2 arrays in the shared memory. Basically, the kernel function in method 1 is split into 2 functions in order to reduce two blocks array of using shared memory. Because horizontal and vertical inverse finite difference can be calculated separately, the original function is split into two Kernel_IFDs, one is for horizontal IFD and the other is for vertical IFD. In each kernel function, only two block arrays are declared in the device shared memory. Method 4 shown in Figure 5.10 illustrates the implementation of two split functions. Similar to method 2, calculating $\left(\mathbf{p}_i - \frac{\lambda_i}{\beta}\right)$ preliminarily allows using only one array block in the shared memory. One can compare method 1 to 4 to see how the number of block arrays declared in shared memory effects the performance. Besides, the effect of reducing bandwidth for loading data from device global memory will be revealed in comparing these methods.



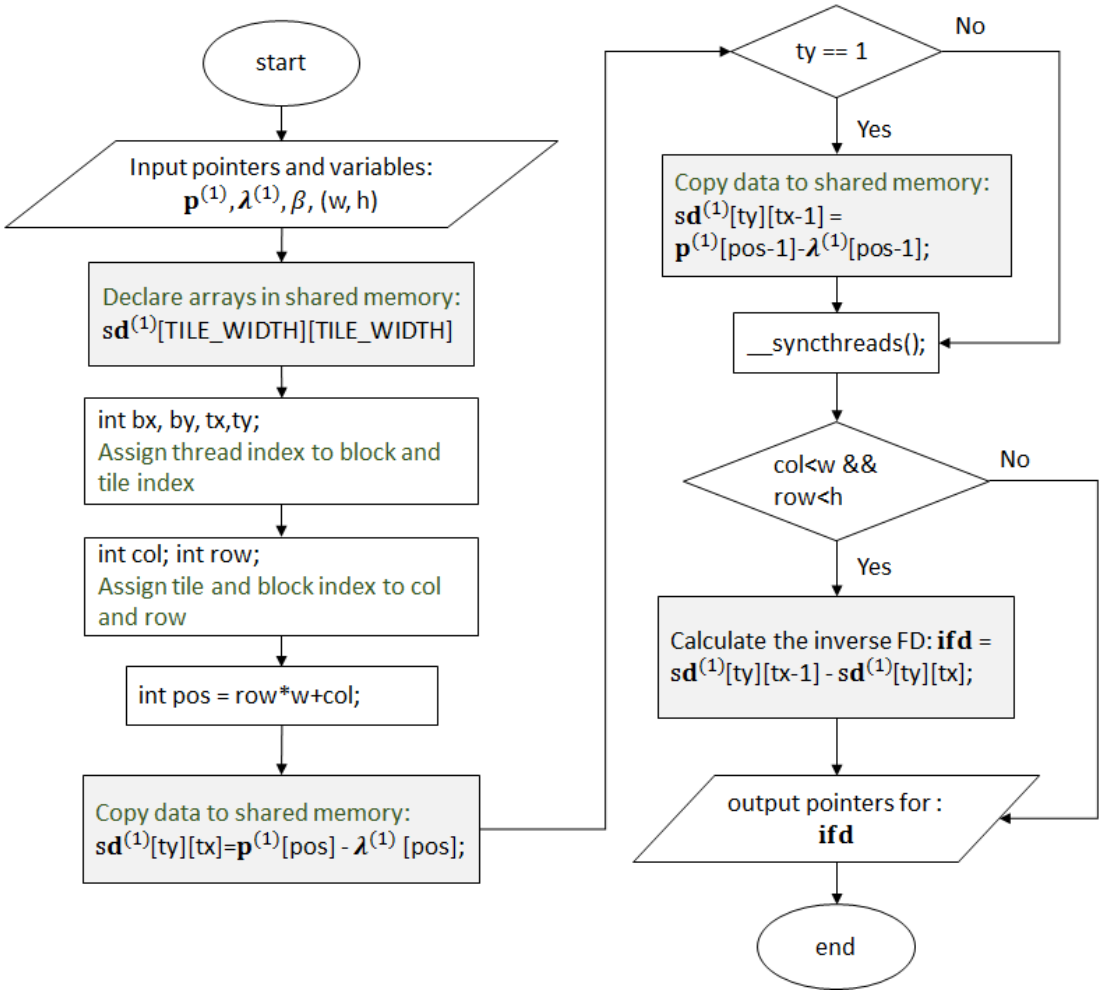
(a)

Figure 5.9 Method 3 - GPU codes of two Kernel functions using 2 arrays in the shared memory for calculating (a) Vertical IFD (b) Horizontal IFD.



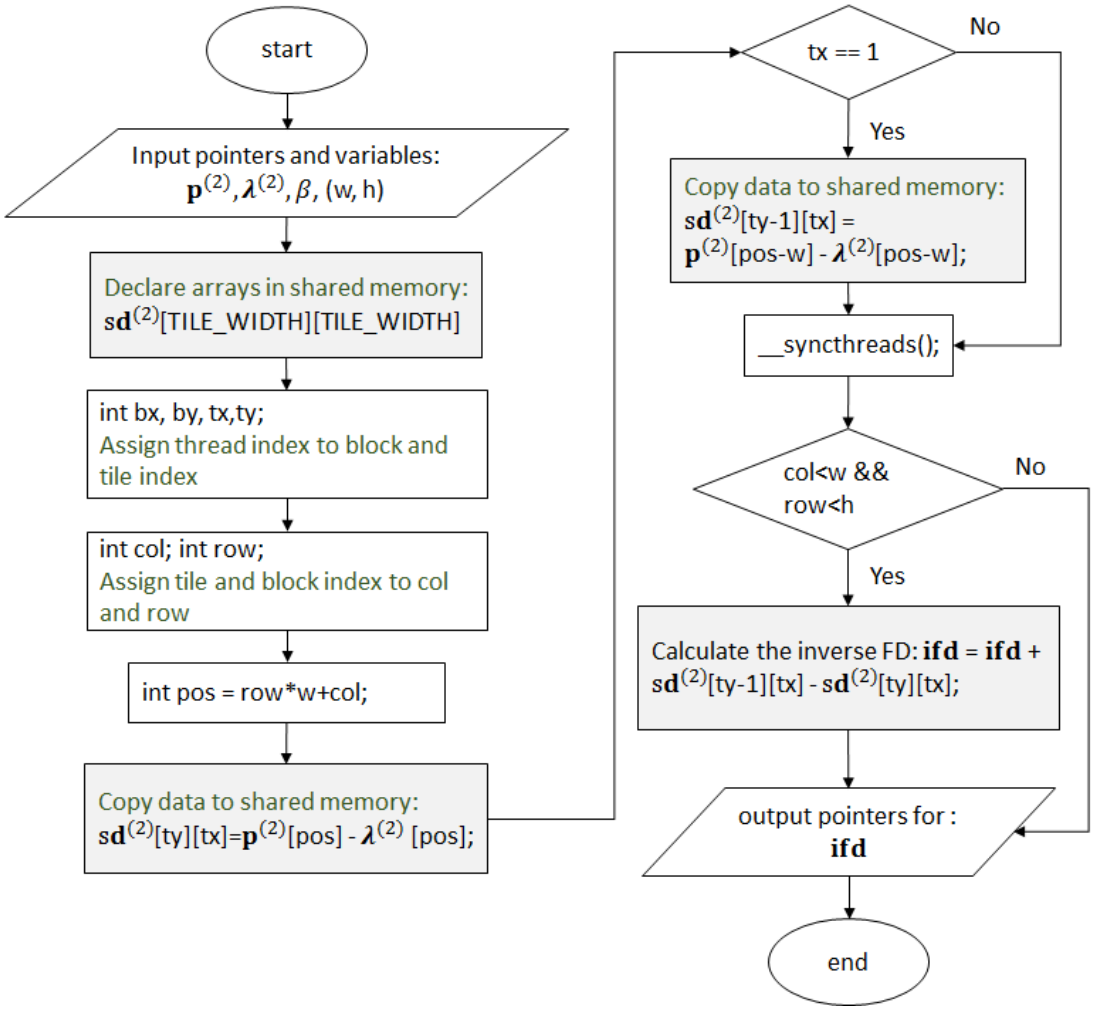
(b)

Figure 5.9 Continued.



(a)

Figure 5.10 Method 4 - GPU codes of two kernel functions using one array in the shared memory for calculating (a) Vertical IFD(b) Horizontal IFD.



(b)

Figure 5.10 Continued.

Data Preparation and Evaluation

To evaluate the performance, both simulated phantom and in-vivo data were used. First, a simulated phantom, which has a volume size of 128×128×16, 8 channel dataset, is used to profile kernel functions of CPU version compared with GPU single precision and double precision. In addition, to test the efficiency of different image size of datasets and the corresponding speedup factors, four-channel datasets with a volume size of 256×256×32,

240×240×32, 128×128×16, 64×64×16, 32×32×4 were synthesized. Three cases are compared: CPU alone, GPU with double precision floating point, and GPU with single precision floating point. Moreover, the four-channel, 256×256×32 dataset is used to test the relation between the iteration number, acceleration and image quality. To evaluate the performance, three measurements are used: (1) Speedup factor, defined as (total runtime on CPU alone)/(total runtime of the proposed method) (2) Normalized mean square errors (NMSEs) = $\frac{\|\mathbf{u}^k - \mathbf{u}_{sos}\|_2}{\|\mathbf{u}_{sos}\|_2}$ (3) Approximation error = $\frac{\|\mathbf{u}_{cpu} - \mathbf{u}_{gpu}\|_2}{\|\mathbf{u}_{cpu}\|_2}$. The entire simulated phantom was down-sampled by random central weighted method along x and y directions. The amount of sampled data was about 16% of a fully-sampled data in all cases. In the last phantom simulation, 256×256×32 dataset is again used to test the image quality using the proposed method compared with the zeros-filled method on various sampling rates.

Moreover, to test the efficiency of reconstructing an in-vivo data, a 3D human knee image acquired with 192×384×34 data matrix from a 4-channel knee coil and a healthy volunteer. The number of iterations is set to 100 and the data was down-sampled by 33%, 25%, and 20% on phase encoding directions. The execution time solely includes the time spending on CS solver, which is running on GPU, compared with the Matlab program running on CPU alone. As for the performance comparisons of utilizing hardware resources, the human knee in-vivo data is used with a 33% sampling rate and the maximum iteration is set to 10. There are two GPU kernel functions, which can be potentially accelerated by utilizing the device shared memory. Two methods for the block of Kernel_FD & Kernel_update_ADMM in Figure 5.4 and Figure 5.5 were compared to see the performance of using the device shared memory. Finally, five methods with/without

using shared memory for Kernel_IFD shown in Figure 5.6 to Figure 5.10 were also compared for analyzing the performance of reducing arrays in shared memory and the bandwidth of loading data from the device global memory.

With our design, the program ran on a platform equipped with an Intel Quadcore i7-3770 3.6 GHz CPU and 32G DDR3 memory. This main work used NVIDIA Geforce GTX 650Ti with a 2G DDR5 memory for parallel CS solver. The Geforce GTX 650Ti has NVIDIA the compute capability 3.0 based on the Kepler GK104 architecture, which consists of 4 streaming multiprocessors (SMX) and each SMX has 192 CUDA cores, total 768 CUDA cores, running at 928MHz. Microsoft window 7, 64-bit operating system, CUDA toolkit version 5.0 are installed. In addition, Matlab version R2012a is installed for dealing with the data loading and the parameter settings of the user interface. The programs for CS solvers are compiled into mex files. Thus, Matlab can launch the reconstruction functions directly.

Results

Table 5.1 shows the kernel profiles under CPU and GPU with double-precision, and GPU with single precision. The runtime was recorded in milliseconds. The number of iteration, k_{max}, was set to 375, which was the largest number of iteration as reconstructing images channel by channel in Figure 5.1(a). Make_data_w_b and Make_data_wt_b were only designed for data arrangement in GPU and they just take small part of the total runtime. The last column describes the speedup factor compared with CPU and GPU single precision. GPU gain significant improvements on all kernel functions. Especially for those

of element-wise operations, such as Kernel_shrinkage and Kernel_IFD, GPU provides larger speed-up factors and has shown a great improvement. Generally, it takes 150 seconds using CPU, 9 seconds using GPU double precision and 4 seconds using GPU single precision.

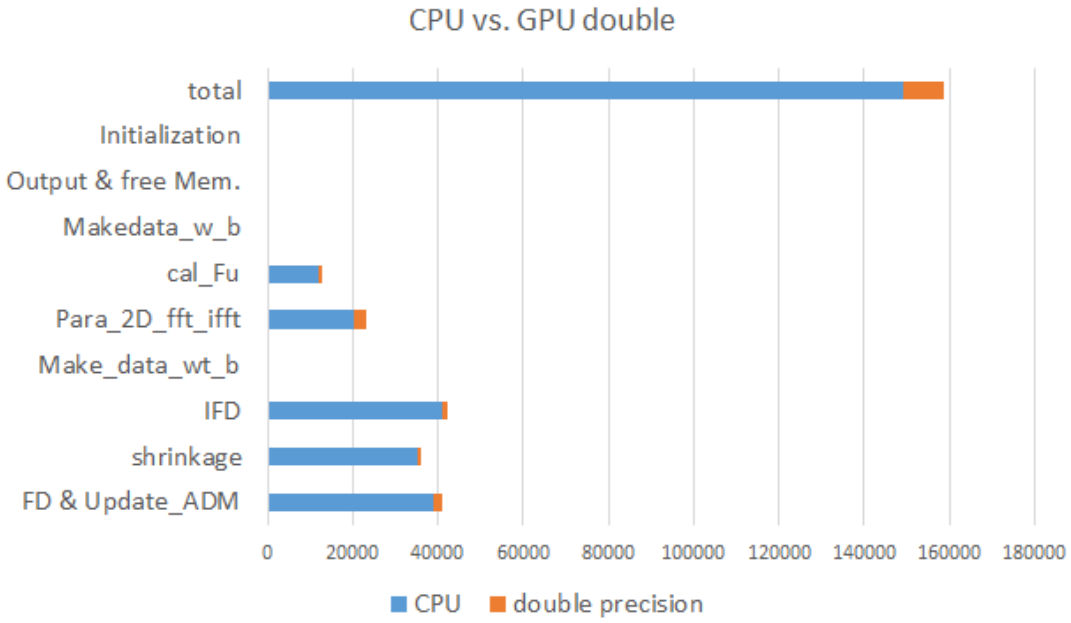
Table 5.1 Runtimes in milliseconds of kernel functions running on GPU, compared with those running on CPU (iteration number, k_{max}=375).

	CPU	GPU(double)	GPU(single)	Speed up
Kernel_FD & Kernel_Update_ADM	38766	2243	1342	28.9
Kernel_shrinkage	34992	1142	566	61.8
Kernel_IFD	41022	1159	808	50.8
Make_data_wt_b	x	467	228	x
Para_2D_fft & Para_2D_ifft	20323	2960	730	27.8
Cal_Fu_i	12127	627	306	38.1
Makedata_w_b	x	552	318	x
Total	149268	9215	4341	34.4

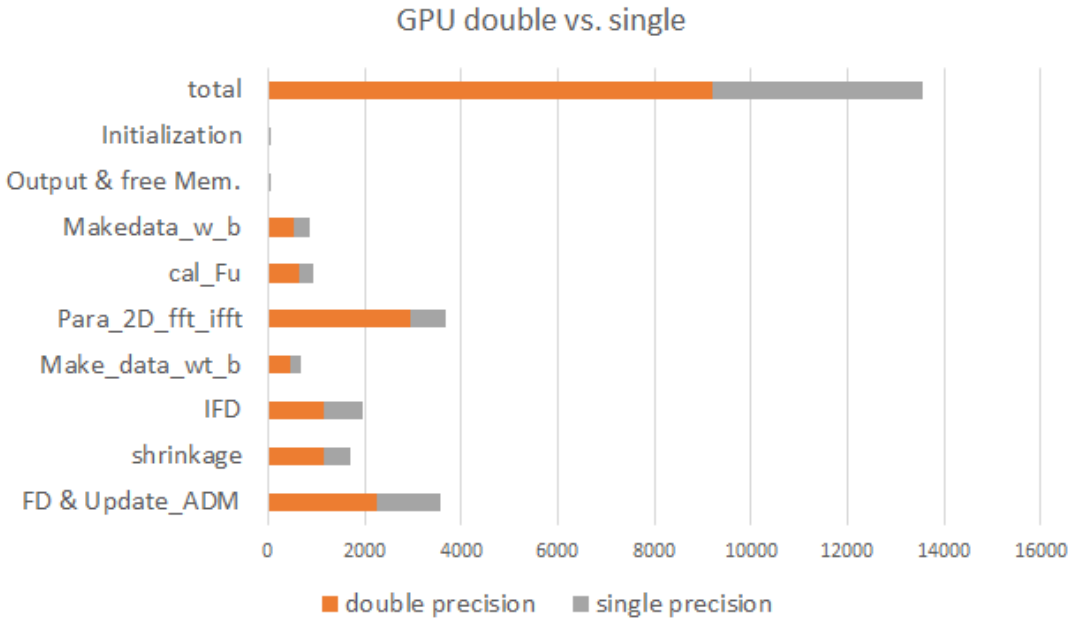
Figure 5.11(a) shows the runtime comparison between CPU and GPU double precision. GPU provides significant reductions in the runtime. The overhead of preparing data for GPU, such as Initialization, Output & free Mem, Make_data_w_b and Make_data_wt_b, only took a small portion of runtime compared with CPU runtime. Figure 5.11(b) shows the runtime comparison between GPU double precision and single

precision. For all kernel functions, it reduced over half of runtime by using single precision. This indicates that the bandwidth of global memory access dominates the runtime.

Figure 5.12(a) illustrates the runtime percentage of kernel functions running on CPU, GPU double precision and GPU single precision. With CPU as shown, Kernel_IDF took the largest allocation; Kernel_shrinkage was the second, and then Kernel_FD & Kernel_update_ADMM" was the third. In CPU codes, these function used for-loop to calculate element-wise operations, which can be highly accelerated by the parallel computing. The second pie chart in Figure 5.12(b) shows the kernel runtimes in percentage using GPU double precision. Parallel 2D FFTs and IFFTs took the largest percentage instead. With GPU double precision, the runtimes on these functions of element-wise operations were significantly reduced because data were highly parallelized and same operations can be simultaneously by GPU cores. With GPU single precision in Figure 5.12(c), Kernel_FD & Kernel_update_ADM took the largest portion. Because launching parallel 2D FFTs and IFFTs consumed memory bandwidth, they gained more improvements when the data were reduced to single precision.



(a)



(b)

Figure 5.11 Runtime comparisons of kernel functions between (a) CPU vs. GPU double precision (b) GPU double precision vs GPU single precision.

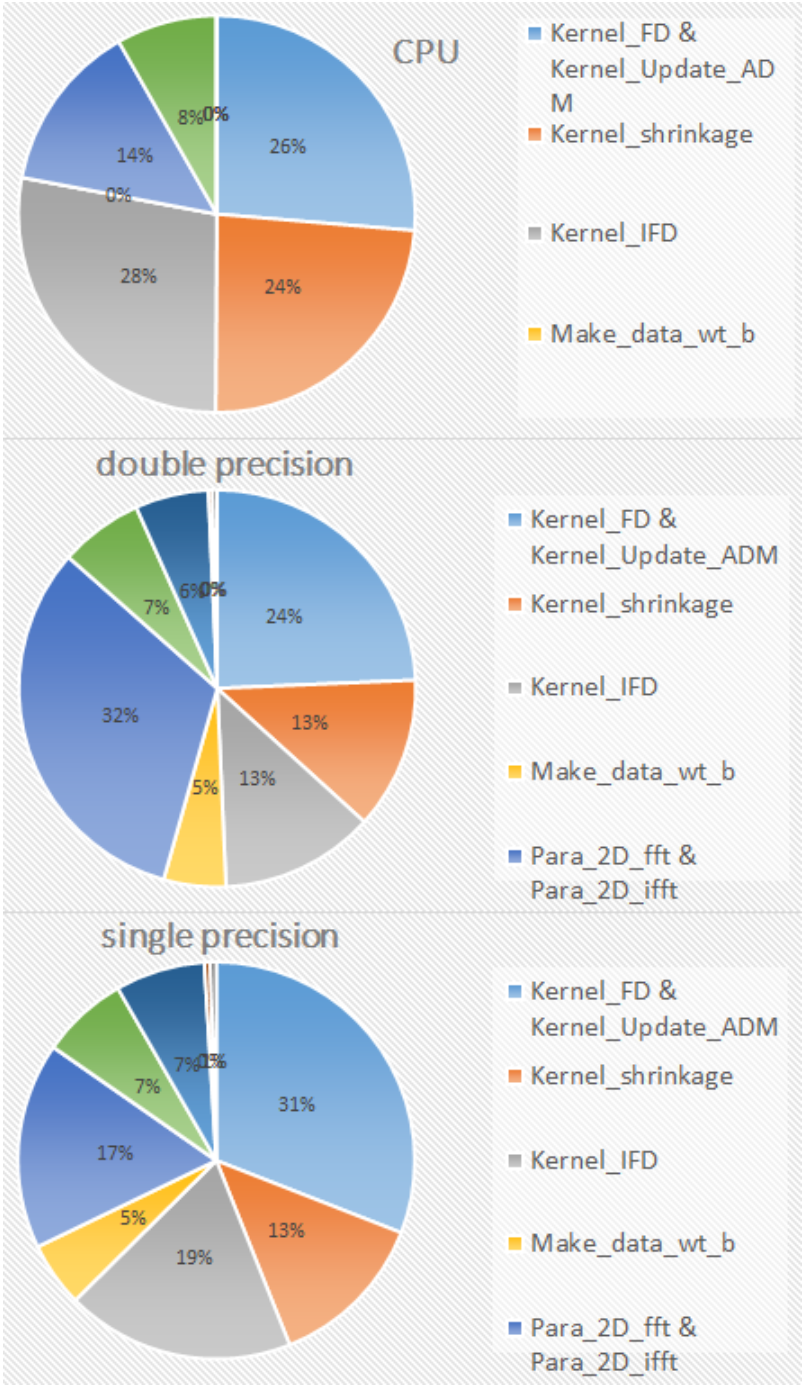


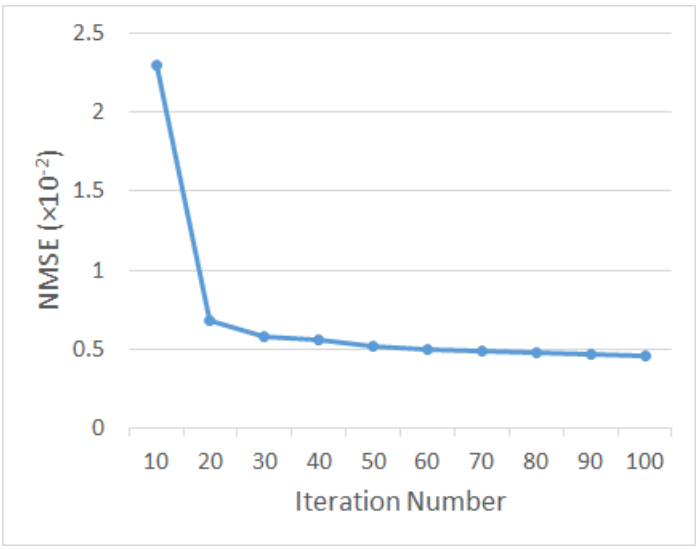
Figure 5.12 Pie charts showing the runtimes of kernel functions in percentage when CPU, GPU double precision, and GPU single precision were used.

Table 5.2 Runtime in milliseconds and speedup factor for a 4-channel dataset with different volume sizes(iteration number, k_{max}=50).

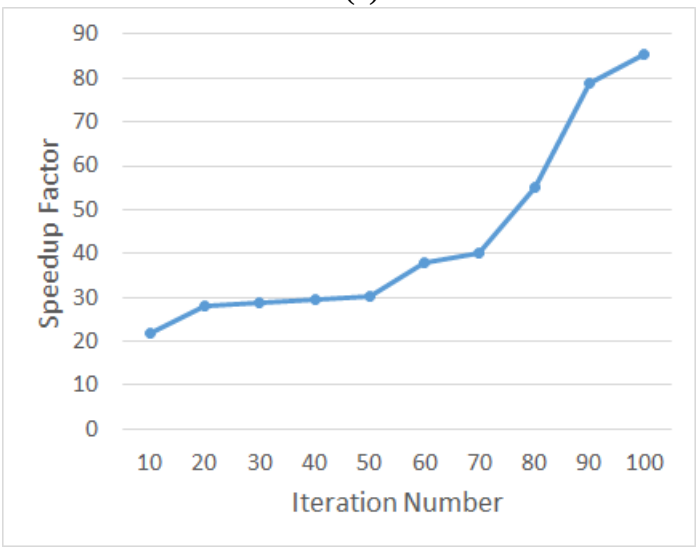
[Nx Ny Nz]	CPU (msec.)	GPU (msec.)	Speed up
256×256×32	67229	2385	28.5
240×240×32	60678	2746	22.1
128×128×16	8795	315	27.9
64×64×16	2497	222	11.2
32×32×4	139	144	0.9

In the next evaluation, Table 5.2 compares the efficiency of the proposed method with CPU implementation for variable dataset size. The stop criteria of the iteration for a 4-channel data with a volume size of 256×256×32 is set to tolerance$< 5.0 \times 10^{-4}$, where k_{max} is about 50. The same iteration number is applied to all datasets for comparisons. For the first dataset, the runtime on GPU was only 2.3 seconds and 28 times faster than single CPU. Intuitively, for the small size dataset, 32×32×4, the overhead of preparing data for GPU, such as Initialization, Output & free Mem, Make_data_w_b and Make_data_wt_b, become more significant. Therefore, the speedup factor is less than 1. For larger datasets, the speedup factor increases because the benefits of data parallelism become more dominant. Note that the speedup factor for the test dataset, 240×240×32×4 (channels), is around 22, which is a little lower than 128×128×16×4(channels) and 256×256×32×4(channels). This is because the dimensional array of threads is set to 16×16. Therefore, the memory access for 240×240×32 dataset is not as efficient as these for the

other two datasets, where the size is multiples of 16. Figure 5.13 illustrates the relation between image quality, reconstruction acceleration and iteration number. The image quality in terms of NMSEs ($\times 10^{-2}$), which is used to compare the reconstruction error with the reference image using SOS method. As shown, the speedup factor increases not only with the size of data, but also with the iteration numbers. In these two experiments, parallelization on GPU gain more improvements on large dataset and higher demand of iterations. Table 5.3 shows the image quality of the reconstruction in terms of NMSEs comparing various sampling rates. The image quality deteriorates when the sampling rate goes down to 8.3%.



(a)



(b)

Figure 5.13 Comparisons of iteration number, image quality and reconstruction acceleration: (a) NMSE ($\times 10^{-2}$) as a function of iteration number; (b) speedup factor as a function of iteration.

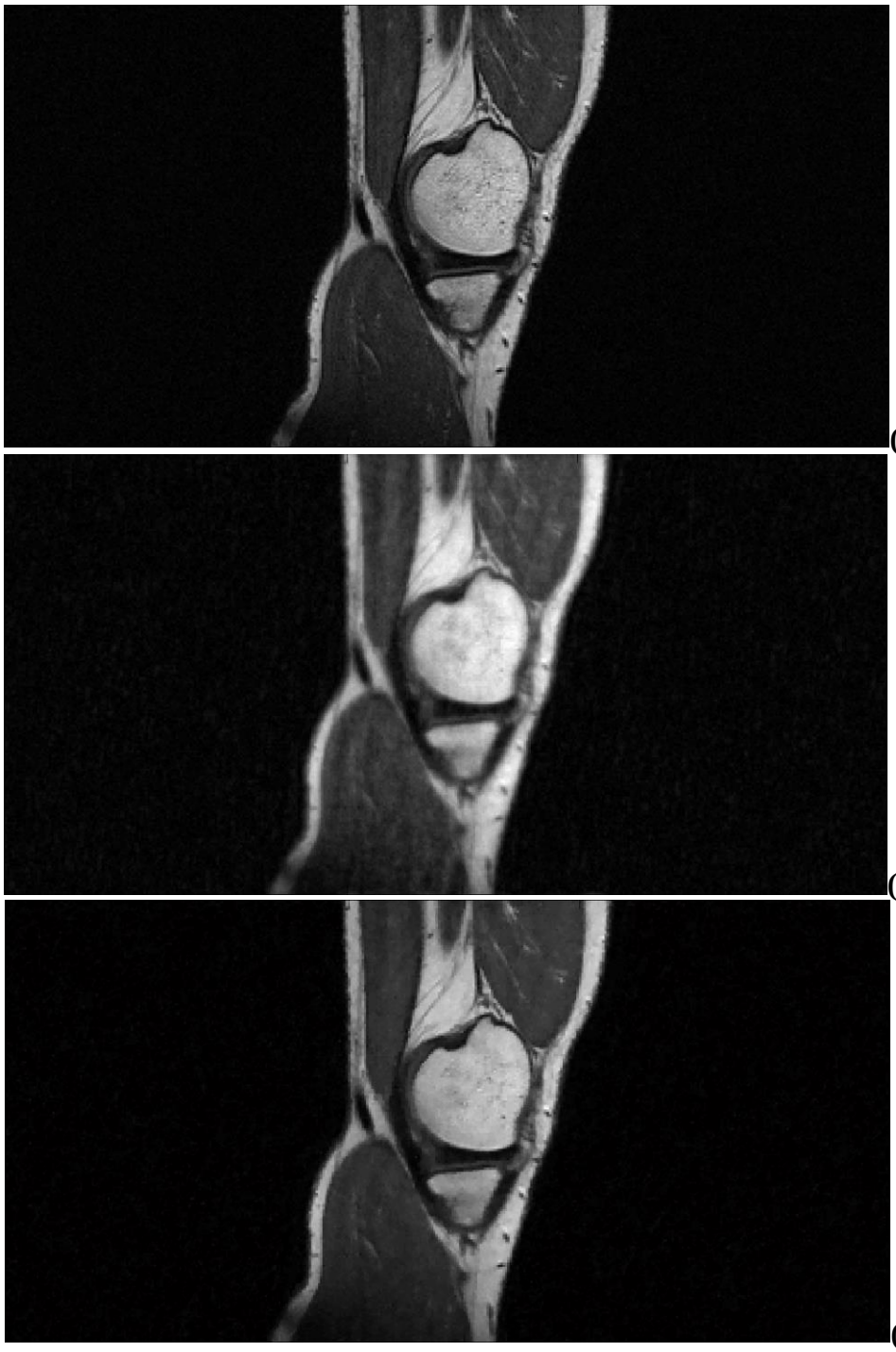


Figure 5.14 Image reconstruction of an in-vivo human knee dataset with (a) SOS (b) zero-filled (c) the proposed method by 33% of original data. It took less than 1 second to reconstruct 34 slices of images with the proposed method.

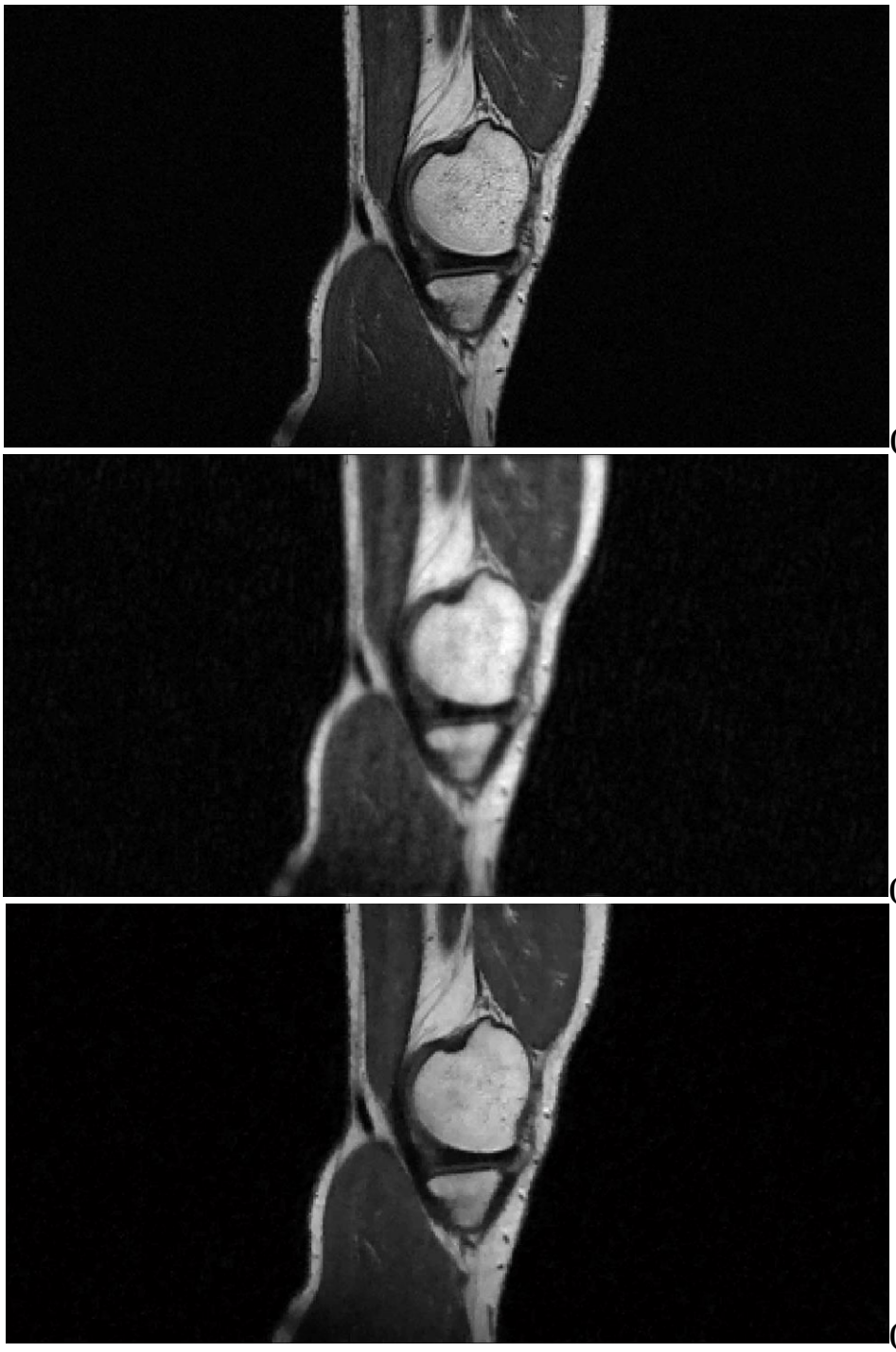


Figure 5.15 Image reconstruction of an in-vivo human knee dataset with (a) SOS (b) zero-filled (c) the proposed method by 25% of original data.

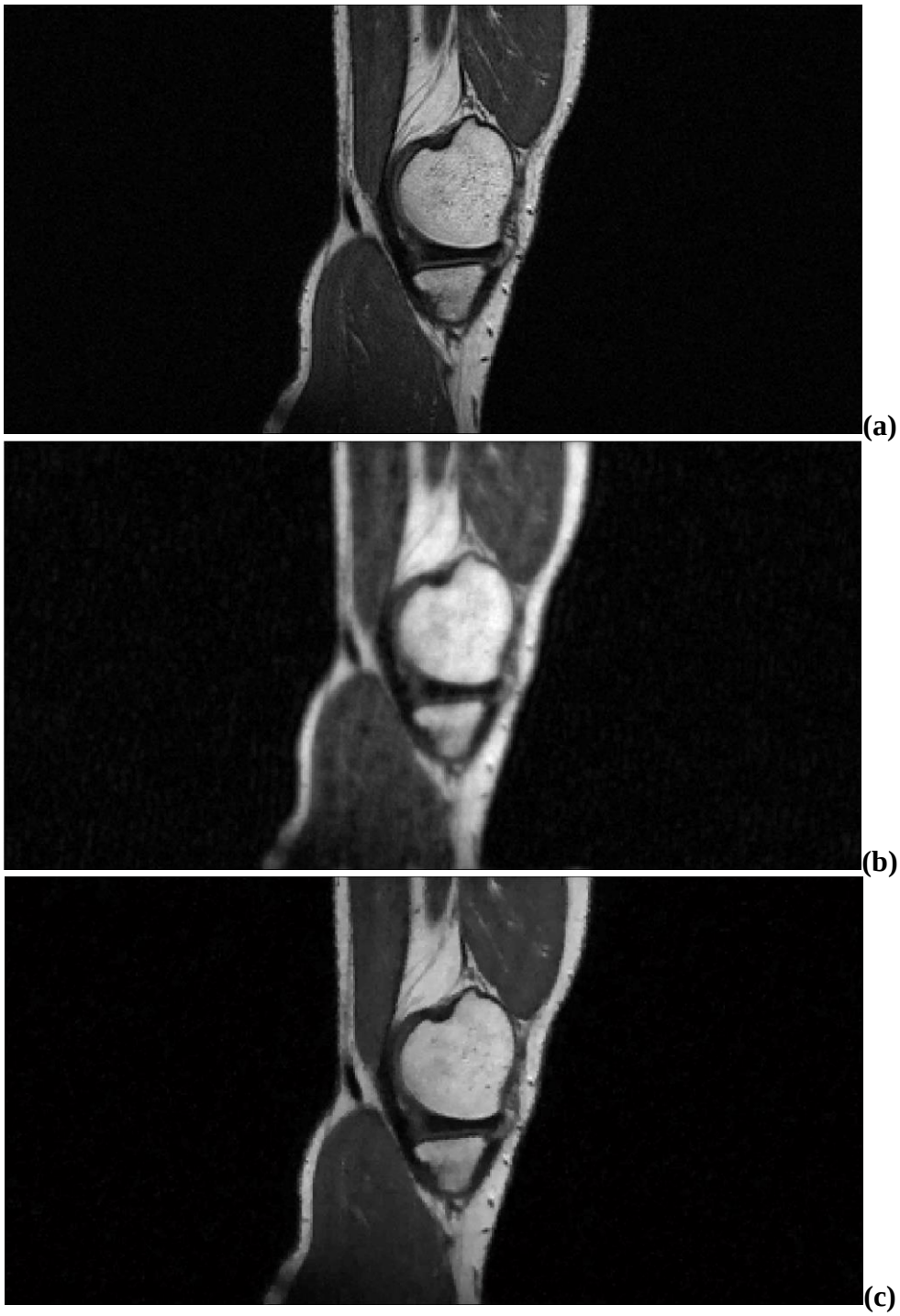


Figure 5.16 Image reconstruction of an in-vivo human knee dataset with (a) SOS (b) zero-filled (c) the proposed method by 20% of original data.

Table 5.3 Comparisons of image quality in terms of NMSEs with various sampling rates using simulated data (iteration number, k_{max}=50).

	25%	16.7%	12.5%	8.3%
Zero-filled	0.3	0.33	0.35	0.36
Proposed	0.3×10^{-2}	0.49×10^{-2}	0.72×10^{-2}	2.1×10^{-2}

Finally, the reconstructed images are presented in Figure 5.14 to Figure 5.16. Figure (a) shows the reconstructions of the SOS method from fully sampled data. Figure 5.14(b) shows the reconstructions of the zero-filled method from 33% of the original dataset, compared with the proposed method in Figure 5.14(c). Figure 5.15(b) and (c) show the comparisons of the zero-filled method and the proposed method from 25% of the original dataset. Figure 5.16(b) and (c) show the results from 20% of the original dataset. The stop criteria for the iterations is set to tolerance$< 3.5 \times 10^{-5}$, where k is about 100. In the last case, the CPU runtime is about 278 seconds while running on GPU only requires 6.7 seconds, making 41 times faster than using CPU alone. The approximation error between the reconstruction results of using CPU and GPU single precision is less than 1.1×10^{-7}. When $k_{\max} = 10$, the runtime on CPU is 19 seconds, and the image quality degrades 1% in terms of $\frac{\|\mathbf{u}_{cpu}{}^{100}-\mathbf{u}_{cpu}{}^{10}\|_2}{\|\mathbf{u}_{cpu}{}^{100}\|_2}$. At this error, there is no image artifacts in the reconstruction. Comparing to 0.95 second on GPU, the speed up factor is about 20. Therefore, in this experiment, it can achieve sub-second reconstruction for a practical 3D knee MRI acquisition.

Performance Considerations

In this section, the usages of the device shared memory, for two GPU kernels are compared. In Geforce GTX 650Ti there is a 64KB configurable L1 and L2 cache for each SMX, where the L1 cache is to cache temporary registers, and L2 cache (shared memory) is to cache accesses from global memory.

Table 5.4 lists the runtime of Kernel_FD & Kernel_update_ADMM with and without the device shared memory. The speed up factor is about 1.2. Since almost every pixel are accessed by three times (except for the boundary pixels), the traffic of accessing the global memory can be significantly reduced. With using shared memory, the execution of each thread block accesses data from a tile block in the shared memory, which requires an array size of 17×17×4(float) ≈ 1.13K bytes. In CUDA compute capability 3.0, the maximum size of the configurable shared memory is 48K bytes, and the maximum number of thread blocks is 16, which can be simultaneously processed in each SMX. Therefore, it requires 1.13K×16 ≈ 18K bytes in total, which is below the capacity of the device shared memory. In this case, the maximum number of thread blocks will not be changed, but the traffic of accessing global memory can be reduced when using shared memory.

Table 5.4 Runtimes in milliseconds for Kernel_FD & Kernel_update_ADMM with and without using the device shared memory(iteration number, k_{max}=10).

	Without shared	With shared	Speedup
Kernel_FD & Kernel_update_ADM	198	164	1.2

Table 5.5 Runtimes in milliseconds for Kernel_IFD with and without using the device shared memory (iteration number, k_{max}=10).

	Without shared	Method 1 (shared)	Method 2 (shared)	Method 3 (shared)	Method 4 (shared)	Speedup
Kernel_IFD	105	105	90	135	127	1.1

Table 5.5 lists the runtime of Kernel_IFD without using shared memory and two different methods using shared memory. Because each element of array variables is accessed twice (except for the boundary), the traffic of accessing global memory could be potentially reduced. In method 1, for the tile size of 17×17, the execution of each thread block accesses four array data, which require 17×17×4(float)×4(arrays) ≈ 4.52K bytes in the shared memory and allow only 10 thread blocks (48/4.52) in maximum being processed concurrently. In this case, the maximum number of thread blocks will be reduced from 16 to 10 in each SMX, which leads to a 3/8 reduction of thread block that can reside in each SMX simultaneously. Conceptually, using the device shared memory can increase the floating point operations per cycle, but the reduction of thread blocks in SMX eliminates the parallelism, which may be the reason that there is no improvement with the method 1 with the shared memory. Comparing to the method 2, the arrays in the shared memory is reduced to two. Each thread block needs 17×17×4(float)×2(arrays) ≈ 2.26K bytes, which reaches the maximum 16 thread blocks (48/2.26≈21). So the reduction in the accesses of global memory contributes to 1.1 increases on the runtime.

In addition, the runtime increased when using method 3 and method 4 in Table 5.5. Though the declaration of tile blocks is reduced from 4 to 2, the function is split into two

functions. One function deals with the horizontal IFD and the other deals with the vertical IFD. The functions are called sequentially, so the number of operations that each thread can process is significantly reduced, which leads to an increasing runtime. Comparing to method 3, two arrays declared in the shared memory are reduced to one in method 4. The usage of one or two arrays in the shared memory makes no difference to the maximum number of thread block in SMX, the number of operations in the function is reduced, which slightly shortens the runtime. However, when the memory is not a limiting factor of parallelism, splitting functions is not a good strategy.

More hardware resources can be utilized for further improvements. For instance, in the left-hand side of eq. (5. 8), $D^T D$ is diagonalized by the 2D discrete Fourier Transform, F. Therefore, horizontal and vertical finite difference operator become element-wise operations, and can be saved as a table in the device memory, such as constant memory or texture memory. This table will be accessed by the GPU kernel function, Cal_F$\mathbf{u}_i$. However, the array table size varies according to different image sizes of datasets, and there is only a 64KB constant memory for each SMX. Thus, it exceeds the capacity of the constant memory for an image size of 256×256. Instead, texture memory will be more suitable hardware resource to promote the runtime performance. In our in-vivo MRI experiment, the GPU kernel, Cal_F$\mathbf{u}_i$, took about 39 millisecond, which is not critical in the final performance. In this case, we will leave the usage of texture memory in the future work.

In addition, to achieve a more aggressive sampling rate, the proposed reconstruction method with GPU can directly fit CS-SENSE, where alias channel images

are independently reconstructed channel-by-channel based on CS, followed by a Cartesian SENSE method. Because Cartesian SENSE is of pixel-wise operations, it can be highly parallelized in GPU without increasing much computational complexity. Besides, in a large array system, coil sensitivities are highly localized. We may assume that channel images are sparser as the number of channel increases. In this case, we may achieve a lower sampling rate and a better SNR in our proposed model. To proof this, more experiments are required on lower sampling rates to test the sparsity of channel images in a large MRI array system. These will be considered as our future work.

Discussion

This paper presented an efficient image reconstruction method of compressed sensing MRI for parallel receive data using GPU. In this method, reconstructions of all channels and slices can be parallelized on the GPU. The reconstruction is based on an efficient alternating direction method of multiplier CS reconstruction for each channel. To balance the bandwidth of accessing global memory and the number of parallel execution threads, several methods to utilize the device shared memory were studied. Experiments and tests with phantom and in-vivo MRI datasets demonstrated that GPU implementation can accelerate image reconstruction by a speedup factor of 25-40. This capability makes it possible to achieve sub-second 3D multi-slice CS reconstruction from array data. In addition, judicial use of the shared memory can further accelerate the GPU implementation of kernel functions by a speedup factor up to 2. The promising results of the study confirm the benefits of GPU for CS-MRI reported in the early literature and

show potentials of enabling CS-MRI reconstructions in real time for future clinical applications.

CHAPTER VI

CONCLUSIONS

Magnetic resonance imaging, providing superior soft-tissue contrast, high spatial resolution and no exposure to ionizing radiation, is a versatile medical imaging technique. However, the slow speed of data acquisition limits its clinical applications. Current up-to-date MRI scanners are equipped with phased array receiver systems, which can speed up the scan time and/or covers a wider area in each scan. As the phased array coils evolve, the imaging approaches and reconstruction methods have been developed, such as well-established parallel imaging. On the other hand, compressed sensing emerged as a powerful theory in data acquisition during the past few years. As a result, it has drawn great attentions in combining compressed sensing with phased array receiver system or exploiting sparsity model in parallel imaging. Although applying CS in MRI acquisition system offers a way to further reduce the number of acquired data, and therefore, shorten the scan time, the cost is in exchanging of longer reconstruction time. CS requires l_1 minimization, which can be solved by iterative numerical algorithms and is often computationally intensive. Therefore, this dissertation mainly focuses on shortening the image reconstructions of compressed sensing MRI with multichannel data using multicore processors.

In chapter I and II, we introduce the background of MRI and the sparsity and incoherence of CS when applied in MRI with multiple coils. A description of data acquisition mechanism in MRI system is briefly presented in chapter I. With phased array receiver system, well-developed parallel imaging methods, such as SENSE and

SPACERIP, are discussed for conventional accelerating data acquisition. The modern theory of CS is described in chapter II. Certain frequently used algorithms for solving l_1 minimization problems of CS recovering, such as iterative shrinkage thresholding and alternating direction method of multipliers, are also presented in order to thoroughly deploy recent research on CS MRI applications. Several CS MRI reconstruction methods are briefly described including sparseSENSE, Self-feeding sparse SENSE, CS-SENSE and l_1 SPIR-iT. This gives a comprehension of current trends on CS MRI reconstructions, which combine with parallel imaging and exploit the sparsity of multi-channel system.

From chapter III through V, we presented a straightforward reconstruction procedure for applying CS in phased array receiver system. From randomly under-sampling channel data, the proposed method independently reconstructs channel images by solving l_1 minimization, and then combines all images via the sum-of-squire method, which provides an asymptotically optimal SNR as the number of channel increases. The best feature of this approach is that channel images are decoupled and can be reconstructed independently and simultaneously. Therefore, it is possible for using multicore processors to accelerate the runtime of image reconstructions. Since the computational complexity scales with the number of channels, parallel computing is implemented to reduce an excessive long time in signal recovery.

In chapter III, we used ubiquitous multi-core CPUs to reconstruct CS images simultaneously. Channel data from different coils were automatically pipelined and processed by different cores of CPU. The proposed reconstruction flow can benefit from executing CS solvers in parallel with multiple cores of CPU, and therefore, the

reconstruction time can be significantly shortened via this sort of parallel computing. In this work, the proposed reconstruction procedure was tested for multi-channel and single slice 2D imaging. According to our experiment results, using 2 cores of CPU gave maximum efficiency improvement per core with respect to the simulated 4-channel data; while using 4 cores gave the fastest reconstruction. However, the efficiency was not doubled as we changed the number of cores from 2 to 4. This is because all cores share the same memory, whose bandwidth and the size are fixed. In addition, the proposed procedure can also fit multi-slice 2D imaging where parallelization is implemented along multiple slices instead of multiple channels. For more computationally intensive cases, such as multi-slice multi-channel imaging or 3D imaging, the efficiency gained from the parallelization will be even more beneficial. In all CS reconstructions studied in this chapter, the overhead is only a small portion of the total runtime. Significant reductions of computational time were achieved using a core 2 quad CPU, especially for higher computation complexity of wavelet transforms using in the l_1 minimization. In this work, the parallelization was implemented with Matlab. However, implementing with C/C++, where improvements will be complementary to the gains achieved by parallelization, can further shorten the computational time. Other existing methods using GPU architecture and more advanced synergetic integration of multi-core CPUs with GPUs can be potentially used to further accelerate CS iterative algorithms for 2D or 3D multi-channel data. These possibilities were explored in chapter V to fully realize the potential of parallel computing for processing large-scale, multi-channel data in MRI.

In chapter IV, a new improved reconstruction method for CS MRI with multi-channel data was presented. In this method, the image is reconstructed using the reweighted l_1 minimization algorithm in a channel-by-channel fashion. Since l_1 norm is an approximation of the sparsity measurements, using reweighted l_1 norm will theoretically give a closer solution to that of the l_0 minimization. The simulated experimental results show that the proposed method can provide an improved image quality comparing to the results without reweighting. The proposed method can also be applied to other CS methods where l_1 minimization is used. However, the new algorithm requires more iterations than the conventional l_1 minimization algorithm. This might pose a problem when immediate delivery of images is preferred. In such cases, using multi-core processors such as graphics processing units (GPUs) can be applied to parallelize the reconstructions and to shorten the reconstruction time.

Although iterative approaches of CS enable the possibility of high reduction factor in MRI scanning and/or an improved image quality, long runtime still poses a barrier to clinical applications. To solve this problem, we presented implementation strategies of CS reconstructions using GPU in chapter IV, which generates results just in few seconds and gain significant runtime improvements for 3D multi-channel data. In our proposed method, channel data and the slices along frequency encode are highly parallelized. Therefore, it substantially reduces reconstruction time by parallel computing with GPU. Generally, for a simulated 4-channel data with a volume size of 256×256×32, the runtime on GPU only requires 2.3 seconds. Comparing to 67 seconds on CPU, it achieves 28 faster for parallel reconstructions. In all experiments, the speedup factors are around 22 to 28,

which can be higher depending on the number of iterations and the size of the dataset. Moreover, with our method, it is needless to acquire more data for estimating coil sensitivities. In addition, it doesn't require gridding algorithm, which may cause some approximation in reconstructions. In this paper, we present the image quality by comparing with the other two traditional methods, recovering by zero-filled and SOS from fully sampled data. Several approaches of using device shared memory are also analyzed by comparing their runtime.

Future work will focus on increasing the reduction factor in MRI scanning since acceleration rate is limited by the sparsity of MR images itself in a channel-by-channel CS reconstruction. To achieve a more aggressive reduction factor, the proposed reconstruction method using GPU can directly fit CS-SENSE, where channel images are independently reconstructed channel-by-channel and its final image is reconstructed pixel by pixel using Cartesian SENSE. In the second step of CS-SENSE, the operations for SENSE are element-wise and can be highly parallelized in GPU, where the computation complexity should fall in the same order of SOS. Therefore, the runtime may not increase too much and these will leave as our future work.

REFERENCES

[1] Z.-P. Liang and P. C. Lauterbur, *Principles of magnetic resonance imaging: a signal processing perspective*. New York: IEEE Press, 2000.

[2] L. Landini, V. Positano, and M. Santarelli, *Advanced image processing in magnetic resonance imaging*. Boca Raton: CRC Press, 2005.

[3] N. B. Smith and A. Webb, *Introduction to medical imaging: physics, engineering and clinical applications*. New York: Cambridge university press, 2010.

[4] J. S. Hyde, A. Jesmanowicz, T. M. Grist, W. Froncisz, and J. B. Kneeland, "Quadrature detection surface coil," *Magnetic Resonance in Medicine,* vol. 4, pp. 179-184, 1987.

[5] J. F. Schenck, H. R. Hart, T. H. Foster, W. A. Edelstein, P. A. Bottomley, R. W. Redington, *et al.*, "Improved MR imaging of the orbit at 1.5 T with surface coils," *American Journal of Neuroradiology,* vol. 6, pp. 193-196, 1985.

[6] M. Hutchinson and U. Raff, "Fast MRI data acquisition using multiple detectors," *Magnetic Resonance in Medicine,* vol. 6, pp. 87-91, 1988.

[7] P. Roemer, W. Edelstein, C. Hayes, S. Souza, and O. Mueller, "The NMR phased array," *Magnetic Resonance in Medicine,* vol. 16, pp. 192-225, 1990.

[8] C. E. Hayes, N. Hattes, and P. B. Roemer, "Volume imaging with MR phased arrays," *Magnetic Resonance in Medicine,* vol. 18, pp. 309-319, 1991.

[9] M. A. Ohliger and D. K. Sodickson, "An introduction to coil array design for parallel MRI," *NMR in Biomedicine,* vol. 19, pp. 300-315, 2006.

[10] J. Ra and C. Rim, "Fast imaging using subencoding data sets from multiple detectors," *Magnetic Resonance in Medicine,* vol. 30, pp. 142-145, 1993.

[11] J. Carlson and T. Minemura, "Imaging time reduction through multiple receiver coil data acquisition and image reconstruction," *Magnetic Resonance in Medicine,* vol. 29, pp. 681-687, 1993.

[12] E. G. Larsson, D. Erdogmus, R. Yan, J. C. Principe, and J. R. Fitzsimmons, "SNR-optimality of sum-of-squares reconstruction for phased-array magnetic resonance imaging," *Journal of Magnetic Resonance,* vol. 163, pp. 121-123, 2003.

[13] M. Blaimer, F. Breuer, M. Mueller, R. M. Heidemann, M. A. Griswold, and P. M. Jakob, "SMASH, SENSE, PILS, GRAPPA: how to choose the optimal method," *Topics in Magnetic Resonance Imaging,* vol. 15, pp. 223-236, 2004.

[14] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger, "SENSE: sensitivity encoding for fast MRI," *Magnetic Resonance in Medicine,* vol. 42, pp. 952-962, 1999.

[15] W. E. Kyriakos, L. P. Panych, D. F. Kacher, C. F. Westin, S. M. Bao, R. V. Mulkern, *et al.*, "Sensitivity profiles from an array of coils for encoding and reconstruction in parallel (SPACE RIP)," *Magnetic Resonance in Medicine,* vol. 44, pp. 301-8, Aug 2000.

[16] M. A. Griswold, P. M. Jakob, M. Nittka, J. W. Goldfarb, and A. Haase, "Partially parallel imaging with localized sensitivities (PILS)," *Magnetic Resonance in Medicine,* vol. 44, pp. 602-609, 2000.

[17] D. K. Sodickson and W. J. Manning, "Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays," *Magnetic Resonance in Medicine,* vol. 38, pp. 591-603, 1997.

[18] M. A. Griswold, P. M. Jakob, R. M. Heidemann, M. Nittka, V. Jellus, J. Wang, *et al.*, "Generalized autocalibrating partially parallel acquisitions (GRAPPA)," *Magnetic Resonance in Medicine,* vol. 47, pp. 1202-1210, 2002.

[19] M. Weiger, K. P. Pruessmann, and P. Boesiger, "2D SENSE for faster 3D MRI," *Magnetic Resonance Materials in Physics, Biology, and Medicine,* vol. 14, pp. 10-19, 2002.

[20] K. P. Pruessmann, M. Weiger, P. Börnert, and P. Boesiger, "Advances in sensitivity encoding with arbitrary k-space trajectories," *Magnetic Resonance in Medicine,* vol. 46, pp. 638-651, 2001.

[21] L. Sha, H. Guo, and A. W. Song, "An improved gridding method for spiral MRI using nonuniform fast Fourier transform," *Journal of Magnetic Resonance,* vol. 162, pp. 250-258, 2003.

[22] A. A. Samsonov, E. G. Kholmovski, D. L. Parker, and C. R. Johnson, "POCSENSE: POCS-based reconstruction for sensitivity encoded magnetic resonance imaging," *Magnetic Resonance in Medicine,* vol. 52, pp. 1397-1406, 2004.

[23] F. H. Lin, K. K. Kwong, J. W. Belliveau, and L. L. Wald, "Parallel imaging reconstruction using automatic regularization," *Magnetic Resonance in Medicine,* vol. 51, pp. 559-567, 2004.

[24] B. Liu, K. King, M. Steckner, J. Xie, J. Sheng, and L. Ying, "Regularized sensitivity encoding (SENSE) reconstruction using Bregman iterations," *Magnetic Resonance in Medicine,* vol. 61, pp. 145-152, 2009.

[25] L. Ying and J. Sheng, "Joint image reconstruction and sensitivity estimation in SENSE (JSENSE)," *Magnetic Resonance in Medicine,* vol. 57, pp. 1196-1202, 2007.

[26] R. Baraniuk, "Compressive sensing," *IEEE Signal Processing Magazine,* vol. 24, pp. 118-121, 2007.

[27] E. J. Candè and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine,* vol. 25, pp. 21-30, 2008.

[28] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, "Compressed Sensing MRI," *IEEE Signal Processing Magazine,* vol. 25, pp. 72-82, 2008.

[29] J. Romberg, "Imaging via compressive sampling," *IEEE Signal Processing Magazine,* vol. 25, pp. 14-20, 2008.

[30] E. Candes and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Problems,* vol. 23, p. 969, 2007.

[31] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*. New York: Cambridge University Press, 2012.

[32] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic Resonance in Medicine,* vol. 58, pp. 1182-95, Dec 2007.

[33] M. Lustig, J. M. Santos, D. L. Donoho, and J. M. Pauly, "kt SPARSE: High frame rate dynamic MRI exploiting spatio-temporal sparsity," in *Proceedings of the 13th Annual Meeting of ISMRM*, Seattle, 2006.

[34] L. Feng, M. B. Srichai, R. P. Lim, A. Harrison, W. King, G. Adluru, *et al.*, "Highly accelerated real-time cardiac cine MRI using k–t SPARSE-SENSE," *Magnetic Resonance in Medicine*, vol. 70, pp. 64-74, 2013.

[35] P. C. Hansen, *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. Philadelphia: Siam, 1998.

[36] D. H. Brooks, G. F. Ahmad, R. S. MacLeod, and G. M. Maratos, "Inverse electrocardiography by simultaneous imposition of multiple constraints," *IEEE Transactions on Biomedical Engineering*, vol. 46, pp. 3-18, 1999.

[37] J. Tropp and S. J. Wright, "Computational methods for sparse solution of linear inverse problems," *Proceedings of the IEEE*, vol. 98, pp. 948-958, 2010.

[38] M. A. Davenport, M. F. Duarte, Y. C. Eldar, and G. Kutyniok, "Introduction to compressed sensing," in *Compressed Sensing: Theory and Applications*, ed New York: Cambridge University Press, 2012, pp. 1-64.

[39] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3397-3415, 1993.

[40] S. Boyd and L. Vandenberghe, *Convex optimization*. New York: Cambridge university press, 2004.

[41] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Mathematical Programming Computation*, vol. 3, pp. 165-218, 2011.

[42] J. A. Tropp and A. C. Gilbert, "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit," *IEEE Transactions on Information Theory*, vol. 53, pp. 4655-4666, 2007.

[43] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, vol. 57, pp. 1413-1457, 2004.

[44] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, pp. 183-202, 2009.

[45] J. M. Bioucas-Dias and M. A. Figueiredo, "A new TwIST: two-step iterative shrinkage/thresholding algorithms for image restoration," *IEEE Transactions on Image Processing*, vol. 16, pp. 2992-3004, 2007.

[46] E. Candes and J. Romberg. (2005). *l1-magic: Recovery of sparse signals via convex programming*. Available: http://users.ece.gatech.edu/justin/l1magic/, accessed in Feb. 2016.

[47] K. Koh, S.-J. Kim, and S. P. Boyd, "An Interior-Point Method for Large-Scale l1-Regularized Logistic Regression," *Journal of Machine Learning Research*, vol. 8, pp. 1519-1555, 2007.

[48] J. Yang and Y. Zhang, "Alternating direction algorithms for l1-problems in compressive sensing," *SIAM Journal on Scientific Computing*, vol. 33, pp. 250-278, 2011.

[49] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, pp. 1-122, 2011.

[50] T. Goldstein and S. Osher, "The Split Bregman Method for L1-Regularized Problems," *SIAM Journal on Imaging Sciences*, vol. 2, pp. 323-343, 2009.

[51] W. Yin, S. Osher, D. Goldfarb, and J. Darbon, "Bregman iterative algorithms for l1-minimization with applications to compressed sensing," *SIAM Journal on Imaging Sciences*, vol. 1, pp. 143-168, 2008.

[52] E. Esser, "Applications of Lagrangian-based alternating direction methods and connections to split Bregman," *CAM report*, vol. 9, p. 31, 2009.

[53] P. Tseng, "Alternating projection-proximal methods for convex programming and variational inequalities," *SIAM Journal on Optimization*, vol. 7, pp. 951-965, 1997.

[54] B. Liu, F. Sebert, Y. Zou, and L. Ying, "SparseSENSE: randomly-sampled parallel imaging using compressed sensing," in *Proceedings of the 16th Annual Meeting of ISMRM, Toronto*, 2008.

[55] K. King, "Combining compressed sensing and parallel imaging," in *Proceedings of the 16th Annual Meeting of ISMRM, Toronto*, 2008.

[56] B. Wu, R. Millane, R. Watts, and P. Bones, "Applying compressed sensing in parallel MRI," in *Proceedings of the 16th Annual Meeting of ISMRM, Toronto*, 2008.

[57] J. X. Ji, Z. Chen, and L. Tao, "Compressed sensing parallel Magnetic Resonance Imaging," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vancouver, 2008, pp. 1671-1674.

[58] D. Liang, B. Liu, J. Wang, and L. Ying, "Accelerating SENSE using compressed sensing," *Magnetic Resonance in Medicine*, vol. 62, pp. 1574-1584, 2009.

[59] D. Liang, K. King, B. Liu, and L. Ying, "Accelerating SENSE using distributed compressed sensing," in *Proceedings of the 17th Annual Meeting of ISMRM*, Honolulu, 2009, p. 377.

[60] C. Prieto, B. Knowles, M. Usman, P. Batchelor, F. Odille, D. Atkinson, *et al.*, "Autocalibrated approach for the combination of compressed sensing and SENSE," in *Proceedings of the 18th Annual Meeting of ISMRM*, Stockholm, 2010, p. 4862.

[61] F. Huang, Y. Chen, W. Yin, W. Lin, X. Ye, W. Guo, *et al.*, "A rapid and robust numerical algorithm for sensitivity encoding with sparsity constraints: Self-feeding sparse SENSE," *Magnetic Resonance in Medicine*, vol. 64, pp. 1078-1088, 2010.

[62] H. She, R. R. Chen, D. Liang, E. V. Dibella, and L. Ying, "Sparse BLIP: BLind Iterative Parallel imaging reconstruction using compressed sensing," *Magnetic Resonance in Medicine*, vol. 71, pp. 645-660, Mar 18 2014.

[63] P. Beatty, K. King, L. Marinelli, C. Hardy, and M. Lustig, "Sequential application of parallel imaging and compressed sensing," in *Proceedings of the 17th Annual Meeting of ISMRM*, Honolulu, 2009, p. 2824.

[64] S. S. Vasanawala, M. T. Alley, B. A. Hargreaves, R. A. Barth, J. M. Pauly, and M. Lustig, "Improved Pediatric MR Imaging with Compressed Sensing," *Radiology*, vol. 256, pp. 607-616, 2010.

[65] M. Lustig, M. Alley, S. Vasanawala, D. Donoho, and J. Pauly, "L1 SPIR-iT: autocalibrating parallel imaging compressed sensing," in *Proceedings of the 17th Annual Meeting of ISMRM*, Honolulu, 2009, p. 379.

[66] S. Vasanawala, M. Murphy, M. Alley, P. Lai, K. Keutzer, J. M. Pauly, *et al.*, "Practical parallel imaging compressed sensing MRI: Summary of two years of experience in accelerating body MRI of pediatric patients," in *2011 IEEE International Symposium on Biomedical Imaging*, 2011, pp. 1039-1043.

[67] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory,* vol. 52, pp. 489-509, 2006.

[68] E. J. Candès, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, pp. 1207-1223, 2006.

[69] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, pp. 1289-1306, 2006.

[70] S. Wright and L. Wald, "Theory and application of array coils in MR spectroscopy," *NMR in Biomedicine,* vol. 10, pp. 394-410, 1997.

[71] C. Zhao, T. Lang, and J. Ji, "Compressed sensing parallel imaging," in *Proceedings of the 16th Annual Meeting of ISMRM,* Toronto, 2008, p. 1478.

[72] L. Marinelli, C. Hardy, and D. Blezek, "MRI with accelerated multi-coil compressed sensing," in *Proceedings of the 16th Annual Meeting of ISMRM*, Toronto, 2008, p. 1484.

[73] D. Liang, B. Liu, and L. Ying, "Accelerating sensitivity encoding using compressed sensing," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vancouver, 2008, pp. 1667-1670.

[74] W. Wiggers, V. Bakker, A. Kokkeler, and G. Smit, "Implementing the conjugate gradient algorithm on multi-core systems," in *Proceedings of the IEEE International Symposium on System-on-Chip* 2007, pp. 1-4.

[75] A. Borghi, J. Darbon, S. Peyronnet, T. F. Chan, and S. Osher, "A simple compressive sensing algorithm for parallel many-core architectures," *Journal of Signal Processing Systems,* vol. 71, pp. 1-20, 2013.

[76] S. S. Stone, J. P. Haldar, S. C. Tsao, W.-m. W. Hwu, B. P. Sutton, and Z.-P. Liang, "Accelerating advanced MRI reconstructions on GPUs," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 1307-1318, 2008.

[77] F. Knoll, M. Unger, F. Ebner, and R. Stollberger, "Real Time Elimination of Undersampling Artifacts using 3D Total Variation on Graphics Hardware," in *Proceedings of the 17th Annual Meeting of ISMRM*, Honolulu, 2009, p. 2838.

[78] C.-H. Chang and J. Ji, "Compressed sensing MRI with multi-channel data using multi-core processors," in *2009 31th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Minneapolis, 2009, pp. 2684-2687.

[79] M. Lustig. (2007). *Sparse MRI*. Available: http://www.eecs.berkeley.edu/~mlustig/Software.html, accessed in Feb. 2016.

[80] J. V. Kepner, *Parallel MATLAB for multicore and multinode computers*. Philadelphia: Society for Industrial and Applied Mathematics, 2009.

[81] C.-H. Chang and J. Ji, "Improved compressed sensing MRI with multi-channel data using reweighted l 1 minimization," in *2010 32th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Argentina, 2010, pp. 875-878.

[82] D. Liang, K. King, B. Liu, and L. Ying, "Accelerating SENSE using distributed compressed sensing," in *Proceedings of the 17th Annual Meeting of ISMRM*, Honolulu, 2009, p. 377.

[83] C.-H. Chang and J. Ji, "Parallel compressed sensing mri using reweighted L1 minimization," in *Proceedings of the 19th Annual Meeting of ISMRM*, Montréal, 2011, p. 2866.

[84] C.-H. Chang and J. X. Ji, "Improving multi-channel compressed sensing MRI with reweighted l1 minimization," *Quantitative Imaging in Medicine and Surgery*, vol. 4, pp. 19-23, 2014.

[85] C.-H. Chang and J. Ji, "Compressed sensing MRI with multichannel data using multicore processors," *Magnetic Resonance in Medicine,* vol. 64, pp. 1135-1139, 2010.

[86] E. J. Candès, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted l1 minimization," *Journal of Fourier Analysis and Applications,* vol. 14, pp. 877-905, 2008.

[87] L. Ying and Z.-P. Liang, "Parallel MRI using phased array coils," *IEEE Signal Processing Magazine,* vol. 27, pp. 90-98, 2010.

[88] K. T. Block, M. Uecker, and J. Frahm, "Undersampled radial MRI with multiple coils. Iterative image reconstruction using a total variation constraint," *Magnetic resonance in medicine,* vol. 57, pp. 1086-1098, 2007.

[89] Y. Wang, "Description of parallel imaging in MRI using multiple coils," *Magnetic Resonance in Medicine,* vol. 44, pp. 495-499, 2000.

[90] E. J. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory,* vol. 52, pp. 489-509, Feb 2006.

[91] R. Otazo, D. Kim, L. Axel, and D. K. Sodickson, "Combination of compressed sensing and parallel imaging for highly accelerated first-pass cardiac perfusion MRI," *Magnetic Resonance in Medicine,* vol. 64, pp. 767-776, 2010.

[92] S. S. Stone, J. P. Haldar, S. C. Tsao, W.-m. Hwu, B. P. Sutton, and Z.-P. Liang, "Accelerating advanced MRI reconstructions on GPUs," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 1307-1318, 2008.

[93] W. Xiao-Long, G. Jiading, L. Fan, F. Maojing, J. P. Haldar, Z. Yue, *et al.*, "Impatient MRI: Illinois Massively Parallel Acceleration Toolkit for image reconstruction with enhanced throughput in MRI," in *2011 IEEE International Symposium on Biomedical Imaging*, 2011, pp. 69-72.

[94] J. Gai, N. Obeid, J. L. Holtrop, X.-L. Wu, F. Lam, M. Fu, *et al.*, "More IMPATIENT: A gridding-accelerated Toeplitz-based strategy for non-Cartesian high-resolution 3D MRI on GPUs," *Journal of Parallel and Distributed Computing,* vol. 73, pp. 686-697, 2013.

[95] T. S. Sorensen, D. Atkinson, T. Schaeffter, and M. S. Hansen, "Real-time reconstruction of sensitivity encoded radial magnetic resonance imaging using a graphics processing unit," *IEEE Transactions on Medical Imaging,* vol. 28, pp. 1974-1985, 2009.

[96] M. S. Hansen, D. Atkinson, and T. S. Sorensen, "Cartesian SENSE and k-t SENSE reconstruction using commodity graphics hardware," *Magnetic Resonance in Medicine,* vol. 59, pp. 463-468, 2008.

[97] D. Kim, J. D. Trzasko, M. Smelyanskiy, C. R. Haider, A. Manduca, and P. Dubey, "High-performance 3D compressive sensing MRI reconstruction," in *2010 32th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Argentina, 2010, pp. 3321-3324.

[98] D. Kim, J. Trzasko, M. Smelyanskiy, C. Haider, P. Dubey, and A. Manduca, "High-performance 3D compressive sensing MRI reconstruction using many-core architectures," *Journal of Biomedical Imaging,* vol. 2011, p. 2, 2011.

[99] M. Murphy, M. Alley, J. Demmel, K. Keutzer, S. Vasanawala, and M. Lustig, "Fast l1-SPIRiT Compressed Sensing Parallel Imaging MRI: Scalable Parallel Implementation and Clinically Feasible Runtime," *IEEE Transactions on Medical Imaging*, vol. 31, pp. 1250-1262, 2012.

[100] S. Nam, M. Akçakaya, T. Basha, C. Stehning, W. J. Manning, V. Tarokh, *et al.*, "Compressed sensing reconstruction for whole-heart imaging with 3D radial trajectories: A graphics processing unit implementation," *Magnetic Resonance in Medicine*, vol. 69, pp. 91-102, 2013.

[101] Q. Li, X. Qu, Y. Liu, D. Guo, Z. Lai, J. Ye, *et al.*, "Accelerating patch-based directional wavelets with multicore parallel computing in compressed sensing MRI," *Magnetic Resonance Imaging*, vol. 33, pp. 649-658, 2015.

[102] B. Hu, X. Ma, M. Joyce, P. Glover, and B. Naleem, "A GPGPU accelerated compressed sensing with tight wavelet frame transform technique for MR imaging reconstruction," in *2012 IEEE International Conference on Imaging Systems and Techniques (IST)*, 2012, pp. 121-125.

[103] D. S. Smith, J. C. Gore, T. E. Yankeelov, and E. B. Welch, "Real-time compressive sensing MRI reconstruction using GPU computing and split Bregman methods," *International Journal of Biomedical Imaging*, vol. 2012, p. 6 pages, 2012.

[104] F. Piccialli, S. Cuomo, and P. De Michele, "A regularized MRI image reconstruction based on hessian penalty term on CPU/GPU systems," *Procedia Computer Science*, vol. 18, pp. 2643-2646, 2013.

[105] Z. Feng, H. Guo, Y. Wang, Y. Yu, Y. Yang, F. Liu, *et al.*, "GPU accelerated high-dimensional compressed sensing MRI," in *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, 2014, pp. 648-651.

[106] J. Li, J. Sun, Y. Song, and J. Zhao, "Accelerating MRI reconstruction via three-dimensional dual-dictionary learning using CUDA," *Journal of Supercomputing*, vol. 71, p. 2381, 2015.

[107] C.-H. Chang and J. X. Ji, "Compressed Sensing MRI with Multichannel Data Using GPUs," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Osaka, 2013, p. 1391.

[108] Y. Junfeng, Z. Yin, and Y. Wotao, "A Fast Alternating Direction Method for TVL1-L2 Signal Reconstruction From Partial Fourier Data," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, pp. 288-297, 2010.

APPENDIX A

Implementations of Reweighted l_1 Minimization

Reweighted $l_1$minimization is modified from the l_1-magic package, which use several algorithms and concepts, including interior point, newton's method, and log-barrier for recovery of sparse signals. This appendix introduces the related algorithms and shows the modification for the reweighted $l_1$minimization.

Interior Point Method and Log-barrier

Consider the following minimization problem.

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_1$$

The objective function contains two terms. One is the least square error and the other is l_1 norm of x, which is regarded as a penalty term. In addition, $A \in R^{m \times n} (m < n)$, $x \in R^n$ and $y \in R^m$. The parameter, λ is a weight between the least square solution and $l_1$norm of x. Because $\|x\|_1$ is not a smooth function, the gradient-based algorithm cannot be applied. Still, the above problem can be reformulated as a constraint minimization problem.

$$\min_x \|Ax - y\|_2^2 + \lambda \sum_{i=1}^{n} u_i$$

$$\text{subject to } -u_i < x_i < u_i,\ i = 1, \dots, n$$

Therefore, it becomes a differential convex quadratic problem with linear inequality constraints. One way to understand interior point method is by adding barrier function.

The above inequality constraints can be reformulated into objective function by an indicator function, $I_+(u) = \begin{cases} 0, u \geq 0 \\ \infty, u < 0 \end{cases}$. Then, the objective becomes as follows,

$$\min f(x,u) = \min_x \|Ax - y\|_2^2 + \lambda \sum_{i=1}^{n} u_i + \sum_{i=1}^{n} I_+(u_i + x_i) + I_+(u_i - x_i)$$

The third term works as penalty if x_i does not satisfy the constraint. The above objective function can be replaced with the following equation.

$$f_t(x,u) = \|Ax - y\|_2^2 + \lambda \sum_{i=1}^{n} u_i + \frac{1}{t} \sum_{i=1}^{n} -(\log(u_i + x_i) + \log(u_i - x_i))$$

When $t \to \infty$, $f_t(x,u)$ is approaching to $f(x,u)$. Therefore, in each iteration given a fixed t, the problem of solving $\min_{x,u} f_t(x,u)$ is equivalent to solve $\min_{x,u} t f_t(x,u)$. The solution, $(x^*(t), u^*(t))$, of $\min_{x,u} t f_t(x,u)$ will be close to the true solution, (x^*, u^*), of $\min_{x,u} f(x,u)$ as t goes to infinity. The parameter t is iteratively increased by $t_{k+1} = \mu t_k$, where the trajectory of varying t is known as central path, and $(x^*(t), u^*(t))$ is called central point. As long as the initial point is inside the interior of the constraints, the solution should always stay inside the interior. One feasible starting point is $x = 0$ and $u = 1$.

Newton's Method

Newton's method in calculus is iteratively finding the roots ($f(x) = 0$) of a differentiable function, $f(x)$. The concept was applied in optimization, which search the solution to $f'(x) = 0$. In each iteration of the above minimization problem, Newton's method can be used to solve the unconstrained subproblem of $\min_{x,u} tf_t(x,u)$. Newton's method is also an iterative method, which solves the problem along $(\Delta x, \Delta u)$ and the search direction satisfies $H\begin{bmatrix}\Delta x \\ \Delta u\end{bmatrix} = -g$, where $H = \nabla^2 f_t(x,u)$ and $g = \nabla f_t(x,u)$. For large-scale problem, it is more economic to solve by the matrix-free conjugate gradient method.

Steepest Descent and Conjugate Gradient

Descent method frameworks are widely used for unconstrained optimization. Basically, they consist of two steps. First, find a descent direction, and then minimize the objective function according to the line along the descent direction. The difference between steepest descent and the conjugate gradient is the choice of descent direction. Before introducing descent method, the gradient of a quadratic form must be clearly defined as $f'(x) = \left[\frac{\partial}{\partial x_1}f(x) \quad \frac{\partial}{\partial x_2}f(x) \quad \dots \quad \frac{\partial}{\partial x_n}f(x)\right]$, which is a vector field. Its meaning of the gradient is that given a point, x, it indicates the direction which has the greatest increase of $f(x)$. Therefore, for steepest descent method (also known as gradient descent), the descent direction is chosen as $-f'(x)$, which implies the direction of greatest decrease of $f(x)$. Therefore, the new point $x_{(1)} = x_{(0)} + \alpha r_{(0)}$ will fall somewhere on the line along $-f'(x)$, where residual $r_{(i)} = -f'(x)$ is equal to the direction of steepest descent. α

denotes a step size, which minimizes $f(x)$ along the line as the directional derivative $\frac{d}{d\alpha}f(x_{(1)})$ is equal to zero. The result becomes as $\frac{d}{d\alpha}f(x_{(1)}) = f'(x_{(1)})^T \frac{d}{d\alpha}x_{(1)} = f'(x_{(1)})^T r_{(0)} = 0$. Thus, one can find the value of α such that $f'(x_{(1)})$ and $r_{(0)}$ are orthogonal. Also, each residual is orthogonal to the previous residual, ${r_{(1)}}^T r_{(0)} = 0$ because $f'(x_{(1)}) = -r_{(1)}$. Note that the convergent path is zigzagging because each descent direction is orthogonal to the previous gradient. If the curve of a quadratic function is smooth and flat, it may converge extremely slowly.

On the other hand, the conjugate gradient can converge at most n steps (where the size of x is equal to n). Assume there are n search directions, $\{d_{(0)}, d_{(1)}, \dots, d_{(n-1)}\}$. The idea is that, in each search step, the length of the step is lined up evenly with one element of x. After n steps, $x_{(i+1)} = x_{(i+1)} + \alpha_{(i)} d_{(i)}$, it converges to the solution, where error,$e_{(i+1)}$ should be orthogonal to $d_{(i)}$. In fact, the matrix A is positive-definite and that makes the contour ellipsoidal. Therefore, instead of making search directions orthogonal, any two directions are A-orthogonal, or conjugate, defined as ${d_{(i)}}^T A d_{(j)} = 0$. Also, $e_{(i+1)}$ requires being A-orthogonal to $d_{(i)}$, which contributes a minimum point along the search direction. According to $\frac{d}{d\alpha}f(x_{(i+1)}) = 0$, α can be calculated. Another part is how to decide the search directions,$\{d_{(i)}\}$ of conjugate gradient method. Assume that there are n linear independent vectors $\{u_i\}$. By applying conjugate Gram-schmidt process, $d_{(i)}$ can be calculated by subtracting components, which are not A-orthogonal to the previous d vector. It turns out $d_{(i)} = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}$, where $i > 0$, and $d_{(0)} = u_0$. The difficulty

is that previous search directions must be kept in memory for calculating the new direction. Replacing $\{u_i\}$ with the $\{r_{(i)}\}$ can solve this problem due to the important properties: the residual $r_{(i+1)}$ is A-orthogonal to the previous search direction except $d_{(i)}$. Therefore, conjugate Gram-Schmidt process becomes easy as $d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$, where $d_{(0)} = r_{(0)}$. It is no longer to save previous search directions, and thus, CG can solve the minimization problem very efficiently.

Nonlinear Conjugate Gradient

Conjugate gradient (CG) is used to solve a linear system and find the minimum of a quadratic form, where system matrix is symmetric, positive-definite. It can also be used to find a minimum of any continuous function, where $f(x)$ is differentiable. There are three major differences between linear and nonlinear CG. First, in linear CG, the recursive residual,$r_{(i+1)}$, is set to $r_{(i)} + \alpha_{(i)} A d_{(i)}$. Instead, the residual of nonlinear case is always set to $r_{(i)} = -f'(x_{(i)})$. Second, the step size, $\alpha_{(i)}$, can be found by $\min_{\alpha} f(x_{(i)} + \alpha_{(i)} d_{(i)})$, which ensures that gradient is orthogonal to the search direction. It is the same as linear CG, and is usually more complicated to compute the step size. Finally, there are several expressions of β, which are equivalent in linear cases. However, different choices of β may lead to different rate of convergence. For example, with Fletcher-Reeves method, $\beta_{(i+1)}^{FR} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$, it converges if the initial point is close to the minimum. With Polak-Ribière method, $\beta_{(i+1)}^{PR} = \frac{r_{(i+1)}^T (r_{(i+1)} - r_{(i)})}{r_{(i)}^T r_{(i)}}$, it often converges much quickly, but could take

infinite steps without converging in rare cases. Adding the condition, , can ensure the convergence. Since CG can only generate n conjugate vector for n iterations for finding the solution, it practically can be restarted every n iterations.

Modified Interior Point Method for Reweighting l_1 Minimization

As CS is applied in MRI applications, the problem can be formulated as

$$\min_x TV(x) \quad \text{s.t.} \quad y = \Phi x$$

where $TV(x) = \sum_{1\le i,j\le n} \left\| D_{ij} x \right\|_2$ and $D_{i,j} x = \begin{bmatrix} D_{h;ij} x \\ D_{v;ij} x \end{bmatrix}$. With this definition, the minimization can be recast as a second order cone problem (SOCP), and solved by log-barrier method. Adding the weights, and the TV minimization becomes

$$\min_x \sum_{1\le i,j\le n} w_{ij}^{(l)} \left\| D_{ij} x \right\|_2 \quad \text{s.t.} \quad y = \Phi x$$

Adding the slack variables, the equation is rewritten as

$$\min_{t,x} \sum_{i,j} u_{ij}$$

$$\text{s.t.} \quad w_{ij}^{(l)} \left\| D_{ij} x \right\|_2 \le u_{ij}$$

$$y = \Phi x$$

Because $w_{ij}^{(l)}$ is chosen according to $\mathrm{w}_{i,j}^{(l+1)} = 1/(\left\| \mathrm{D}_{ij} x^{(l)} \right\|_2 + \varepsilon)$, it is positive, and the problem doesn't change if the inequality is divided by $w_{ij}^{(l)}$.

$$\min_{t,x} \sum_{i,j} u_{ij}$$

$$\text{s.t.} \quad \left\| D_{ij} x \right\|_2 - w_{ij}^{(l)^{-1}} u_{ij} \le 0$$

$$y = \Phi x$$

Let $t_{ij} = {w_{ij}^{(l)}}^{-1} u_{ij}$, and the problem is rewritten as

$$\min_{t,x} \sum_{i,j} w_{ij}^{(l)} t_{ij}$$

$$\text{s.t.} \quad \left\| D_{ij} x \right\|_2 - t_{ij} \leq 0$$

$$y = \Phi x$$

Thus, it becomes obvious to put in a standard form as follows,

$$\min \langle \begin{bmatrix} 0 \\ w \end{bmatrix}, \begin{bmatrix} x \\ t \end{bmatrix} \rangle$$

$$\text{s.t.} \quad f_{t_{i,j}}(z) = \frac{1}{2} (\left\| D_{i,j} x \right\|_2^{\ 2} - t_{i,j}^2) \leq 0$$

$$[\Phi \quad 0] \begin{bmatrix} x \\ t \end{bmatrix} = y, \quad i,j = 1, \dots, n$$

where the inequality functions, $f_{t_{i,j}}$, describes a second-order conic, $c_0 = \begin{bmatrix} 0 \\ w \end{bmatrix}$, $z = \begin{bmatrix} x \\ t \end{bmatrix}$, $A_0 = [\Phi \quad 0]$. The log-barrier method transforms the inequality constraints into the objective function, in which log-barrier performs as a penalty function. When the constraints are violated, the objective become infinite.

$$f_0(z) = \min \langle c_0, z \rangle + \frac{1}{\tau^k} \sum_{i,j} - \log(-f_{t_{i,j}}(z))$$

$$\text{s.t.} \quad A_0 z = y$$

As τ^k gets large, the solution z^k to the above equation approaches to the optimized solution z^*. For each iteration k of log-barrier method, the subproblem can be solved by Newton's method within a few iterations. Then, the solution z^k is used as a starting point for next sumbproblem, $k + 1$. According to second order Taylor expansion of f_0 around z.

$$f_0(z+\Delta z) \approx f_0(z) + \langle g_z, \Delta z\rangle + \frac{1}{2}\langle H_z \Delta z, \Delta z\rangle$$

where $g_z = c_0 + \frac{1}{\tau}\sum_{i,j} \frac{1}{-f_{t_{i,j}}(z)} \nabla f_{t_{i,j}}(z)$, and $H_z = \frac{1}{\tau}\sum_{i,j} \frac{1}{f_{t_{i,j}}(z)^2} \nabla f_{t_{i,j}}(z) \nabla f_{t_{i,j}}(z)^T + \frac{1}{\tau}\sum_{i,j} \frac{1}{-f_{t_{i,j}}(z)} \nabla^2 f_{t_{i,j}}(z)$. Newton's method solves the above equation by setting the derivative with respective to Δz equal to zero. Assume that v represents the Lagrange multipliers for the equality constraint $A_0 z = y$, the minimum can be found along direction Δz given that z is a feasible set to $A_0 z = y$. The search direction satisfies the following equation,

$$\tau \begin{bmatrix} H_z & {A_0}^T \\ A_0 & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ v \end{bmatrix} = -\tau \begin{bmatrix} g_z \\ 0 \end{bmatrix}$$

For the problem of recovering images from noisy observation, there is no equality condition. Instead, the data fidelity term can be added into the objective function using log-barrier.

$$\min \langle \begin{bmatrix} 0 \\ w \end{bmatrix}, \begin{bmatrix} x \\ t \end{bmatrix} \rangle$$

$$\text{s.t.} \quad f_{t_{i,j}}(z) = \frac{1}{2}(\left\| D_{i,j} x \right\|_2^{\,2} - t_{i,j}^2) \leq 0$$

$$f_\varepsilon(z) = \frac{1}{2}(\|Ax - b\|_2^{\,2} - \varepsilon^2) \leq 0$$

Since there is no equality constraint ($A_0 = 0$), the search direction satisfies the equation.

$$\tau H_z \Delta z = -\tau g_z$$

Because adding weights only affects the elements of the vector c_0, only g_z needs to be modified respectively. Note that the above equation is reformulated as

$$\begin{bmatrix} H_{11} & B\Sigma_{12} \\ \Sigma_{12}B^T & \Sigma_{22} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta t \end{bmatrix} =$$

$$-\begin{bmatrix} {D_h}^T F_t^{-1} D_h x + {D_v}^T F_t^{-1} D_v x + F_\varepsilon^{-1} A^T (Ax - b) \\ -\tau w^{(l)} - F_t^{-1} t \end{bmatrix}$$

The other details of equations are described in the package of $l_1$magic notes. The system H_z is symmetric, positive-definite. Thus, it can be solved by CG for large-scale data. As soon as the search direction $\begin{bmatrix} \Delta x \\ \Delta t \end{bmatrix}$ is found, $z = \begin{bmatrix} x \\ t \end{bmatrix}$ is updated by $\begin{bmatrix} x \\ t \end{bmatrix} = \begin{bmatrix} x \\ t \end{bmatrix} + s \begin{bmatrix} \Delta x \\ \Delta t \end{bmatrix}$, where the step size s is decided by backtracking line search.

APPENDIX B

Codes for GPU Implementations

Kernel_FD and Kernel_updata_ADM without using shared memory:

```
__global__ void ComputeKernel_Ux_Uy_bx_by(VolumeType *pUx,
VolumeType *pUy, VolumeType *pbx, VolumeType *pby,
        VolumeType *pU, VolumeType *pWx, VolumeType *pWy, double
gamma, int width, int height)
{///without using shared memory//
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;

    int right;
    int down;
    int pos;

    if(col<width-1 && row<height-1)
    {//avoid accessing outside memory
        right = row * width + (col+1);
        pos = row * width + col;
        down = (row+1) * width + col;


        pUx[pos] = pU[down]-pU[pos];
        pUy[pos] = pU[right]-pU[pos];

        pbx[pos] += gamma*(pUx[pos] - pWx[pos]);
        pby[pos] += gamma*(pUy[pos] - pWy[pos]);
    }
}
```

Kernel_FD & Kernel_updata_ADMM using shared memory:

```
__global__ void ComputeKernel_Ux_Uy_bx_by(VolumeType *pUx,
VolumeType *pUy, VolumeType *pbx, VolumeType *pby,
        VolumeType *pU, VolumeType *pWx, VolumeType *pWy, double
gamma, int width, int height)
{
    __shared__ VolumeType sU[TILE_WIDTH][TILE_WIDTH];


    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x; int ty = threadIdx.y;

    int col = bx * blockDim.x + tx;
    int row = by * blockDim.y + ty;
```

```
    int pos;
    VolumeType tmpx, tmpy;

    pos = row * width + col;

    sU[ty][tx] = pU[pos];
    if(tx==blockDim.x-1)
    sU[ty][tx+1] = pU[pos+1];
    if(ty==blockDim.y-1)
    sU[ty+1][tx] = pU[pos+width];
    __syncthreads();

    if(col<width-1 && row<height-1)
    {//avoid accessing outside memory
        tmpx = sU[ty+1][tx]-sU[ty][tx];
        tmpy = sU[ty][tx+1]-sU[ty][tx];

        pUx[pos] = tmpx;
        pUy[pos] = tmpy;

        pbx[pos] += gamma*(tmpx - pWx[pos]);
        pby[pos] += gamma*(tmpy - pWy[pos]);
    }
}
```

Kernel_IFD without using the shared memory:

```
__global__ void ComputeKernel_rhs(VolumeType *prhs, VolumeType *pWx,
VolumeType *pWy, VolumeType *pbx, VolumeType *pby, float tau, int
width, int height)
{
    int col = blockIdx.x * blockDim.x + threadIdx.x + 1;
    int row = blockIdx.y * blockDim.y + threadIdx.y + 1;

    int left;
    int up;
    int pos;

    if(col<width && row<height)
    {
        pos = row * width + col;
        left = row * width+(col-1);
        up = (row-1) * width + col;

        prhs[pos] = tau*( pWx[up] - pbx[up] - pWx[pos] + pbx[pos] \
                                + pWy[left] - pby[left] - pWy[pos] +
pby[pos] );
    }
}
```

Kernel_IFD using the shared memory Method 1:

```
__global__ void ComputeKernel_rhsUpLeft(VolumeType *prhs, VolumeType
*pWx, VolumeType *pWy, VolumeType *pbx, VolumeType *pby, float tau,
int width, int height)
{
    __shared__ VolumeType sWx[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sbx[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sWy[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sby[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;

    sWx[ty][tx] = pWx[pos];
    sbx[ty][tx] = pbx[pos];
    sWy[ty][tx] = pWy[pos];
    sby[ty][tx] = pby[pos];

    if(ty==1){
    sWx[ty-1][tx] = pWx[pos-width];
    sbx[ty-1][tx] = pbx[pos-width];
    }
    if(tx==1){
    sWy[ty][tx-1] = pWy[pos-1];
    sby[ty][tx-1] = pby[pos-1];
    }
    __syncthreads();

    if(col<width && row<height)
    {
        prhs[pos] = tau*( sWx[ty-1][tx] - sbx[ty-1][tx] -
sWx[ty][tx] + sbx[ty][tx] \
                + sWy[ty][tx-1] - sby[ty][tx-1] - sWy[ty][tx] +
sby[ty][tx] );
    }
}
```

Kernel_IFD using the shared memory Method 2:

```
__global__ void ComputeKernel_rhsUpLeftS1(VolumeType *prhs,
VolumeType *pWx, VolumeType *pWy, VolumeType *pbx, VolumeType *pby,
float tau, int width, int height)
{
    __shared__ VolumeType sDTx[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sDTy[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;

    sDTx[ty][tx] = pWx[pos]-pbx[pos];
    sDTy[ty][tx] = pWy[pos]-pby[pos];

    if(ty==1){
    sDTx[ty-1][tx] = pWx[pos-width]-pbx[pos-width];
    }
    if(tx==1){
    sDTy[ty][tx-1] = pWy[pos-1]-pby[pos-1];
    }
    __syncthreads();


    if(col<width && row<height)
    {
        prhs[pos] = tau*( sDTx[ty-1][tx] - sDTx[ty][tx] \
                + sDTy[ty][tx-1] - sDTy[ty][tx] );
    }

}
```

Kernel_IFD using the shared memory Method 3:

```
__global__ void ComputeKernel_rhsUp(VolumeType *prhs, VolumeType
*pWx, VolumeType *pbx, float tau, int width, int height)
{
    __shared__ VolumeType sWx[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sbx[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;

    sWx[ty][tx] = pWx[pos];
    sbx[ty][tx] = pbx[pos];
    if(ty==1){
    sWx[ty-1][tx] = pWx[pos-width];
    sbx[ty-1][tx] = pbx[pos-width];
    }
    __syncthreads();


    if(col<width && row<height)
    {
        prhs[pos] = tau*( sWx[ty-1][tx] - sbx[ty-1][tx] -
sWx[ty][tx] + sbx[ty][tx] );
    }
}
```

```
__global__ void ComputeKernel_rhsLeft(VolumeType *prhs, VolumeType
*pWy, VolumeType *pby, float tau, int width, int height)
{
    __shared__ VolumeType sWy[TILE_WIDTH][TILE_WIDTH];
    __shared__ VolumeType sby[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;
```

```
sWy[ty][tx] = pWy[pos];
    sby[ty][tx] = pby[pos];
    if(tx==1){
    sWy[ty][tx-1] = pWy[pos-1];
    sby[ty][tx-1] = pby[pos-1];
    }
    __syncthreads();

    if(col<width && row<height)
    {
        prhs[pos] = prhs[pos] + tau*( sWy[ty][tx-1] - sby[ty][tx-1]
- sWy[ty][tx] + sby[ty][tx] );
    }
}
```

Kernel_IFD using the shared memory Method 4:

```
__global__ void ComputeKernel_rhsUpS1(VolumeType *prhs, VolumeType
*pWx, VolumeType *pbx, float tau, int width, int height)
{
    __shared__ VolumeType sDTx[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;

    sDTx[ty][tx] = pWx[pos]-pbx[pos];
    if(ty==1){
    sDTx[ty-1][tx] = pWx[pos-width]-pbx[pos-width];
    }
    __syncthreads();

    if(col<width && row<height)
    {
        prhs[pos] = tau*( sDTx[ty-1][tx] - sDTx[ty][tx] );
    }
}
```

```
__global__ void ComputeKernel_rhsLeftS1(VolumeType *prhs, VolumeType
*pWy, VolumeType *pby, float tau, int width, int height)
{
    __shared__ VolumeType sDTy[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x + 1; int ty = threadIdx.y + 1;

    int col = bx * blockDim.x + tx ;
    int row = by * blockDim.y + ty ;

    int pos;
    pos = row * width + col;

    sDTy[ty][tx] = pWy[pos]-pby[pos];
    if(tx==1){
    sDTy[ty][tx-1] = pWy[pos-1]-pby[pos-1];
    }
    __syncthreads();

    if(col<width && row<height)
    {
        prhs[pos] = prhs[pos] + tau*( sDTy[ty][tx-1] - sDTy[ty][tx]
);
    }
}
```