HIGH PERFORMANCE TOMOGRAPHY

Thesis · August 2017

CITATIONS

0

1 author:

Xiao Wang

Harvard Medical School

7 PUBLICATIONS **9** CITATIONS

SEE PROFILE

HIGH PERFORMANCE TOMOGRAPHY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Xiao Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2017

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Samuel Midkiff, Co-Chair

School of Electrical and Computer Engineering

Dr. Charles Bouman, Co-Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Milind Kulkarni

School of Electrical and Computer Engineering

Approved by:

Dr. V. Ragu Balakrishnan

Head of the School of Electrical and Computer Engineering

To God, who gives me a passionate heart toward research.

Through His natural laws, He allows me to see a universe marvelously arranged.

ACKNOWLEDGMENTS

I would like to thank Professor Samuel Midkiff and Professor Charles Bouman for their support and discreet guidance in my research. It is a great blessing to have their help in my graduate school years. I would also like to thank Professor Anand Raghunathan and Professor Milind Kulkarni for serving on my advisory committee and giving me research ideas.

In addition to the committee members, I would like to thank Dr. Larry Drummy from Air Force Research Lab for providing the iron hydroxide dataset, Dr. Dilworth Parkinson from Lawrence Berkeley National Lab for providing the slime mold dataset, and Dr. Francisco De Carlos from Argonne National Lab for providing the biology dataset, used in Chapter 1. The iron hydroxide dataset was cleared on 11 Jan 2017 by the 88th Air Base Wing public affairs office, case number 88ABW-2017-0095. In addition, I would like to thank the National Energy Research Scientific Computing Center (NERSC) for providing supercomputing resources.

This dissertation was funded by the U.S. Department of Homeland Security under SBIR Award D15PC00110, subcontracted through High Performance Imaging LLC, and by the Indiana Economic Development Corporation (IEDC). Additional support was provided by the DHS ALERT Center for Excellence supported by the U.S. Department of Homeland Security, Science and Technology Directorate, Office of University Programs, under Grant Award 2013-ST-061-ED0001. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security or the IEDC.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

β	rotation angle
r	scanner detector channel
$p_\beta(r)$	projection
y	Computed Tomography measurements
x	voxel
A	forward system matrix of the scanner geometry
D	diagonal weighting matrix
$R(x)$	regularizing prior function
Δ_d	detector channel spacing
Δ_x	voxel width
$\tilde{S}$	the voxel trace average absolute slope in the sinogram
$m(\phi)$	the voxel trace average absolute slope in the Super-Voxel Buffer
Δ_β	view angle spacing in radian
N_v	the number of views in the sinogram
N_{wh}	the number of voxels on each side of a square Super-Voxel
N_d	the number of voxels along the depth of a Three Dimensional Super-Voxel
N_x	the width and height of the image space
$\tilde{A}$	the average amplitude for a voxel trace in the sinogram
$\tilde{A}_{SVB}$	the average amplitude for a voxel trace in the Super-Voxel Buffer
L_{SVB}	the average Super-Voxel Buffer width
L_{pw}	the average voxel trace width
$\delta_1(\beta)$	the length of projection on the X-ray detector at angle β
V_c	the volume of a chunk in a Block Transposed Buffer

E_c the percentage of essential measurements in a Block Transposed Buffer

ABBREVIATIONS

CT	Computed Tomography
MRI	Magnetic Resonance Imaging
PET	Positron Emission Tomography
SV	Super-Voxel
SVB	Super-Voxel Buffer
ICD	Iterative Coordinate Descent Algorithm
GCD	Grouped Coordinate Descent Algorithm
PSV-ICD	Parallel Super-Voxel ICD Algorithm
MBIR	Model-Based Iterative Reconstruction
FBP	Filtered-Back Projection
RMSE	Root Mean Square Error
GFLOPs	Billions of Floating Point Operations Per Second
3DSV	Three-Dimensional Super-Voxel
TIMBIR	Time-Interlaced MBIR
NU-PSV	Non-Uniform Parallel Super-Voxel Algorithm
EDS	Explosive Detection System

ABSTRACT

Wang, Xiao Ph.D., Purdue University, August 2017. High Performance Tomography.
Major Professor: Charles A. Bouman and Samuel P. Midkiff.

Computed Tomography (CT) Image Reconstruction is an important technique used in a wide range of applications, ranging from explosive detection, medical imaging to scientific imaging. Among available reconstruction methods, Model Based Iterative Reconstruction (MBIR) produces higher quality images and allows for the use of more general CT scanner geometries than is possible with more commonly used methods. The high computational cost of MBIR, however, often makes it impractical in applications for which it would otherwise be ideal.

This dissertation describes the concept of the super-voxel (SV) that significantly reduces the computational cost of MBIR while retaining its benefits. It describes how scanner data can be organized into SVs that dramatically increase locality and prefetching, regularize data access pattern, enable massive parallelism, and ensure fast convergence. Experimental results indicate that the super-voxel algorithm has an average speedup of 9776 compared to the fastest state-of-the-art 3D image reconstruction implementation on a 69632-core distributed system.

1. INTRODUCTION

Modern imaging systems increasingly use computation to form useful images from raw sensor data, and this integration of computation and sensing in computational imaging systems is among the most important trends in commercial, scientific, medical and security imaging.

One of the classic example of such a computational imaging system is computed tomography. Computed Tomography (CT) is an imaging technique used to reconstruct a 3D volume by slices, where a slice is defined as a cross-section of the sample object with a fixed thickness, from a set of projections through the use of any kind of penetrating wave. Although CT was primarily used in radiology in the early years [1], CT is now also widely used in archeology [2], biology [3–5], atmospheric science [6], geophysics [6, 7], plasma physics [8], materials science [9–12], quantum information [13], and security baggage scanning [14–18].

Driven by clinical demands and scientific experiments, the CT system hardware design has advanced significantly in the past 30 years. From a historical perspective, parallel-beam and fan-beam CT were the dominant CT system design for years because of their simple design. In later years, the clinical demand for have higher isotropic spatial resolution has driven forth the development of the cone-beam CT systems. With CT's popularity in hospitals, CT-induced radiation exposure becomes a great concern for patients because radiation has enough energy to damage DNA and cause cancer. Therefore, the demand for low radiation doses has become the current trend for CT system design.

Another trend for CT system design is the demand for sparse-view reconstruction. For time-sensitive scientific experiments, the long delay time in collecting datasets for tomographic reconstruction makes some experiments impossible. The only solution for such experiments is to collect a sample set of data and reconstruct an image from

this sample. Without doubt, reconstruction from a sub-sampled dataset, also known as sparse-view reconstruction, is challenging for CT image reconstruction methods.

In parallel with the rapid advances in CT system hardware design, image reconstruction methods, i.e., the algorithms to process CT system measurements and reconstruct an image (or volume) from measurements, has also advanced rapidly. In general, reconstruction techniques fall into two categories: traditional analytical methods, such as filtered back projection (FBP), and model-based iterative reconstruction methods (MBIR). The traditional analytical methods are easy to implement and fast to compute. For these reasons, the traditional analytical methods are the most commonly used reconstruction methods in various applications. In response to the demand for low doses and sparse-view reconstruction, however, the traditional analytical methods for reconstruction are unsuitable and produce undesirable image quality. Therefore, MBIR was developed to meet these needs [4, 5] and to provide much higher image quality.

In comparison to the traditional reconstruction methods, MBIR has allowed dramatic reductions in X-ray dosage for a variety of clinical applications including lung cancer screening (80% reduction) [19] and pediatric imaging (30-50% reduction) [20]. In addition to its popularity in clinical applications, MBIR has also shown to result in higher quality images than other reconstruction methods in applications, including explosive detection systems (EDS) [14–17], medical imaging [4, 5], scientific and materials imaging [10] by significantly improving image spatial resolution, noise as well as artifacts. Figure 1.1 shows the image quality comparison for a sample slice from a fast-evolving biology dataset with very low radiation doses [21]. To show the image quality difference, Figure 1.1 uses both FBP and MBIR reconstruction methods. Figure 1.1(a) is a slice reconstructed by traditional reconstruction method (FBP) and Figure 1.1(b) is the same slice reconstructed by MBIR. We observe that the reconstruction using MBIR has much higher spatial resolution and much less noise than the other reconstruction.

To achieve superior image quality, MBIR, which mathematically is a Bayesian estimation process, requires a careful modeling of the CT system optics and geometries (also known as the "forward model" in Bayesian estimation) and a modeling of the reconstructed image (also known as the "prior model"). The purpose of the forward model is to fit a reconstruction to the CT system measurements while the purpose of the prior model is to preserve edges in images and avoid excessive noise reduction. Because of the mathematical intractability and computational demands, MBIR is formulated as a cost function that incorporates both the forward and prior models, and an optimization method is used to iteratively and monotonically reduce the cost function. With a large enough number of iterations, the reconstructed image will converge to an estimate near the minimizer of the cost function.

While a consensus has emerged that MBIR algorithms have the potential to greatly improve image quality for low-dose and sparse-view reconstructions, MBIR has a long delay time in image reconstruction because of its high computation cost. In typical applications, iterative reconstruction methods may require a factor of 10 to over 100 times the computation of FBP. Consequently, the slow computation speed remains a major barrier for the adoption of MBIR in real-time CT system applications. In a clinical environment, for example, a real-time image reconstruction is often necessary because a radiologist needs an image reconstruction to be completed shortly after the CT scan completes.

Broadly speaking, there are two computational approaches to MBIR: the global update methods [17, 22–24] and the local update methods. The global update methods are perhaps the most widely used optimization approaches, and include gradient descent [17], conjugate gradient [23], preconditioned gradient [23], and conjugate gradient descent [24]. All of these global update methods work by iteratively projecting an entire image into the measurements or *sinogram space*[1]. In addition, the global update methods are easy to implement in parallel because these methods simultaneously update all voxels, where a voxel is defined as a 3D pixel that represents a single data

[1]The measurement or sinogram space stores all measurements collected by CT scanner system.

point on the final image whose value corresponds to the data being reconstructed, e.g., how opaque the portion of the image represented by the voxel is to x-rays.

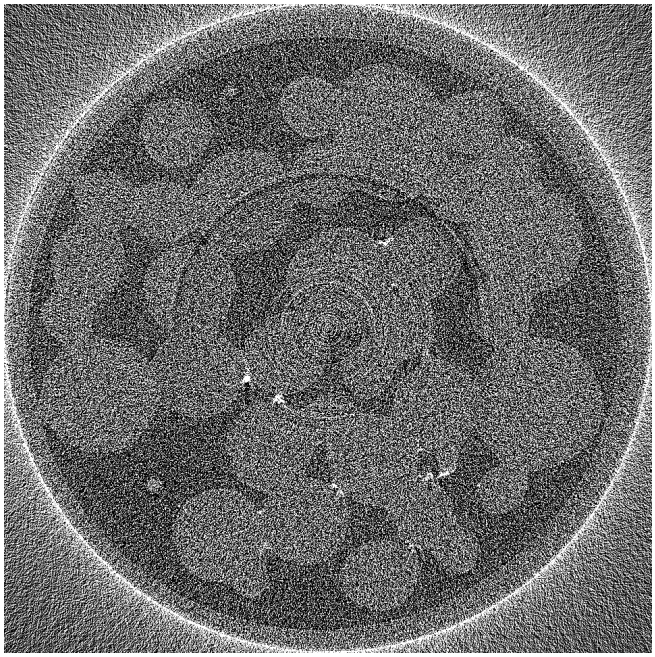
Although the global update methods are suitable for parallelism, they possess a number of disadvantages that slow down computations. One of the disadvantages of the global update methods is their slow algorithmic convergence, which often require hundreds of iterations to converge [17]. To speed up convergence, these algorithms can be combined with methods such as preconditioning [23], ordered subsets [25], or domain decomposition [26]. However, all these methods have significant drawbacks. For example, the preconditioning methods must be custom designed for each CT imaging system and geometry, which require large amounts of manual efforts.

The second approach is the local update method, such as Iterative Coordinate Descent (ICD) [4, 27] and Non-Homogeneous Iterative Coordinate Descent (NHICD) [28], which have been shown to have very rapid and robust numerical convergence for a wide variety of MBIR applications with various geometries, noise statistics, and image models. Intuitively, ICD is fast and robust because each voxel is updated to best minimize the cost function, so there is tight feedback in the optimization process. In practice, the local update methods have been shown to converge in 3 to 6 iterations instead of the hundreds of iterations needed by global update methods [17, 27]. However, while the local update methods have rapid convergence, it is believed that ICD requires operations that are more difficult to parallelize [29,30] because voxel updates depend on each other and parallel voxel updates have significant parallel overhead (see Section 3.4).

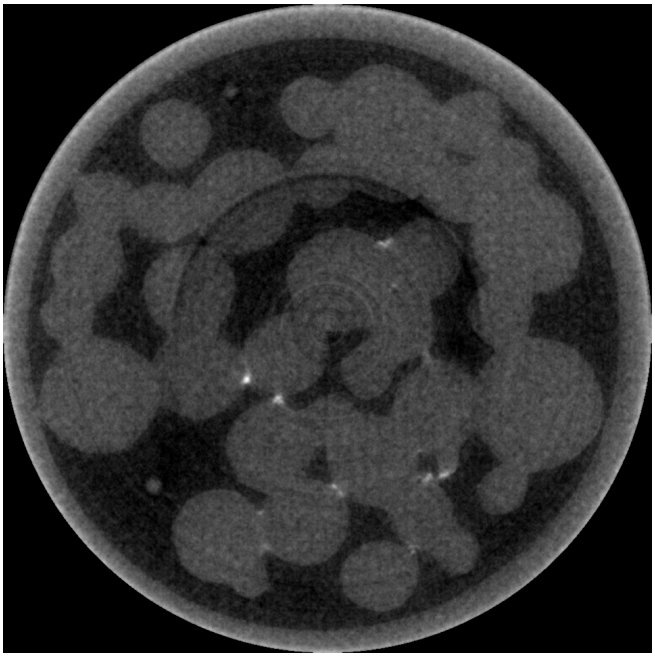
More importantly, all MBIR reconstruction approaches suffer from *poor* cache locality in the data layout. For each voxel's update, measurements have to be accessed following an irregular pattern [31], which in turn worsens the cache locality. To fundamentally address MBIR's computation barriers, all problems discussed above have to be addressed.

This dissertation discusses the performance challenges for MBIR and presents solutions. In particular, it shows how to achieve high level of parallelism, good single-core

performance and fast algorithmic convergence for MBIR algorithms. In Chapter 2, the dissertation will review the CT system setup for 2D and 3D image reconstruction. Chapter 2 will also relate the performance challenges with the nature of the CT system setup. In Chapter 3, the dissertation discusses the performance challenge for 2D image reconstruction. In addition, it introduces the concepts of the *super-voxels* (SVs) and *super-voxel buffer* (SVB) to increase cache locality and hardware prefetching. Then, it discusses how to parallelize SV updates using multi-cores of a computer workstation in Chapter 3. Next, it discusses the performance challenge for fully 3D image reconstruction in Chapter 4. Then, the dissertation presents *three-dimensional super-voxels* (3DSV) to enable SIMD vectorization and regularize access pattern. In addition, the dissertation introduces the Non-Uniform Parallel Super-Voxel Algorithm (NU-PSV) in Chapter 4 and maps this algorithm onto a massively parallel supercomputer. Finally, conclusions and future work are presented in Chapter 6.



(a)



(b)

Fig. 1.1.: (a) A sample slice reconstructed by traditional reconstruction method. (b) The same sample slice reconstructed by MBIR.

2. BACKGROUND

Before we go in depth into how this dissertation addresses the technical problems in high performance image reconstruction, we need to first understand how the Computed Tomography (CT) system is set up and how the setup relates to the performance challenges. In Section 2.1, we will discuss the CT system setup for 2D image reconstruction and the inherent cache locality, and parallel computing challenges. In Section 2.2, we will extend the CT system setup to fully 3D image reconstruction and explain how a fully 3D reconstruction relates to SIMD performance challenges.

2.1 2D COMPUTED TOMOGRAPHY

In this section, we will discuss a typical parallel beam X-ray CT system setup. For simplicity, we assume that the sample object to be scanned has only one slice. Figure 2.1 (a), (b) and (c) illustrate the basic concepts of a parallel beam X-ray CT system. An X-ray source is mounted on a rotating gantry along with a rotating linear array of X-ray detectors. The object to be imaged is then placed at or near the center of rotation. When the gantry containing the X-ray source and sensor rotates, X-rays pass through the object to be imaged and a series of measurements are taken on the array of X-ray detectors. Each single view is taken at a specific rotation angle, β. At that view angle, the CT system takes a set of measurements from all the elements of the X-ray detector array at a single instant of time. Since the X-rays pass through the object from the source to the sensor array, measurements from each array element r (or channel) are used to estimate $p_\beta(r)$, the integral density of the object along the path from the X-ray source to the detector. The objective is then to take these estimates of integral density and reconstruct an image of the object. Figure 2.1 (a), (b) and (c) also show how a CT system takes measurements for the voxels "u_1" and

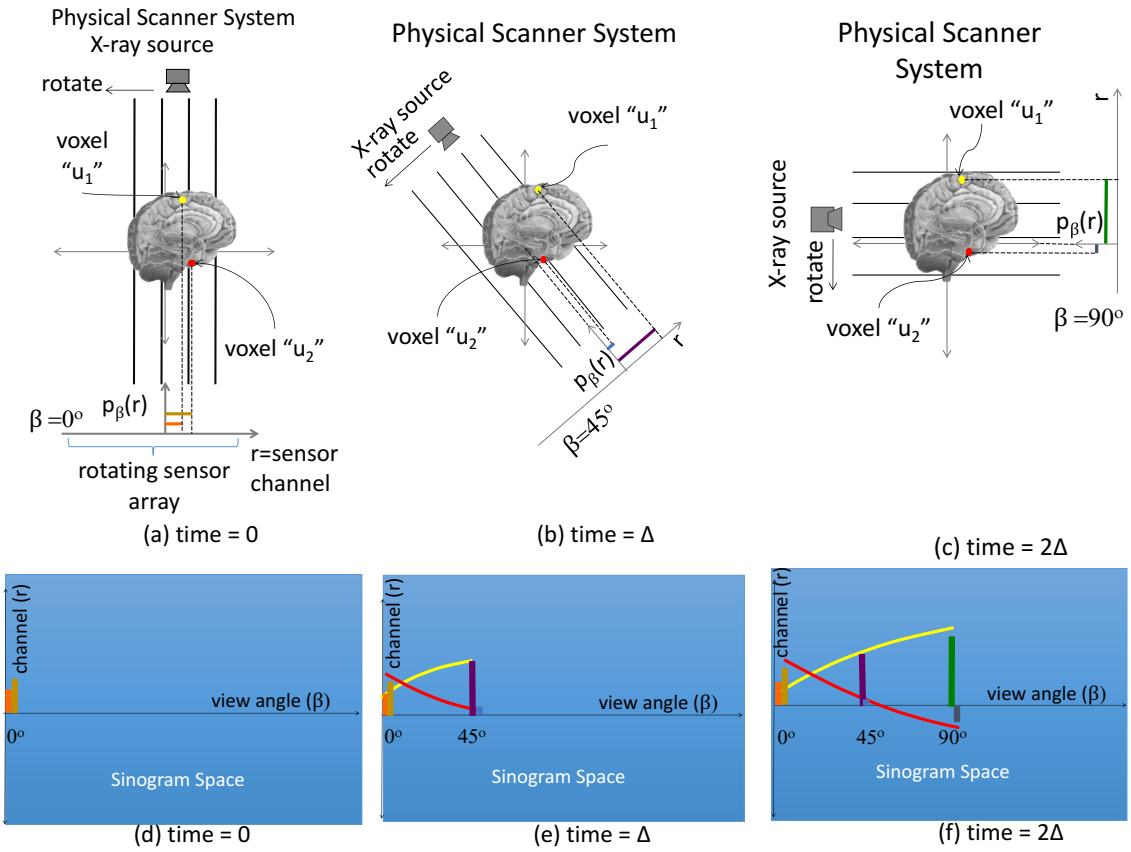


Fig. 2.1.: (a), (b) and (c) illustrate CT tomography system operations for voxels "u_1" and "u_2" when $\beta = 0°$, 45°and 90°respectively. (d), (e) and (f) show how the measurement data collected from the CT scanner is organized into the sinogram.

"u_2" when $\beta = 0°$, 45°and 90°respectively. Bars of different color show the distance between the channel that detects the voxel and the channel located at the center of rotation.

Typically, the data collected from the scanner is organized into a sinogram, as illustrated in Figure 2.1 (d), (e) and (f). For each cross-section slice of the object being imaged, the sinogram is a 2D data structure indexed by the channel r and view β. The name "sinogram" comes from the fact that the measurements of individual voxels in the image correspond to sine wave patterns in the sinogram. When one connects channels r at different views β for voxels "u_1" and "u_2", their connected traces correspond to the sine wave patterns in the sinogram shown as a thin yellow line and a red line. In order to compute the update of a voxel in ICD, it is necessary to access voxels' corresponding measurement values in the sinogram following these sine wave patterns.

Modern processors access main memory by first transferring blocks of local memory onto fast on-chip cache memory that is much faster than main memory. Cache lines are shown as short blue line segments in Figure 2.2. Notice that, for the most part, these horizontal blue cache lines only partially overlap the yellow or red line of memory that must be accessed. This means that most of the space in a cache line does not hold data for the current voxel update.

One additional observation is that each individual voxel corresponds to a sine wave with a different amplitude and phase. Since each voxel has a unique sinusoidal path, no single transformation of the sinogram data will align the cache lines with the sinusoidal paths in all cases. Also, the sinusoidal paths for different voxels will always overlap at some views, shown as blue dots in Figure 2.2. Therefore, any parallel algorithm that updates different voxels simultaneously needs to atomically update these intersection points. When the number of voxels to be simultaneously updated increases, there will be more intersections and more lock contention. This lock contention can limit parallel performance. For this reason, it is not surprising that grouped coordinate decent (GCD) [29, 30], the most commonly known method

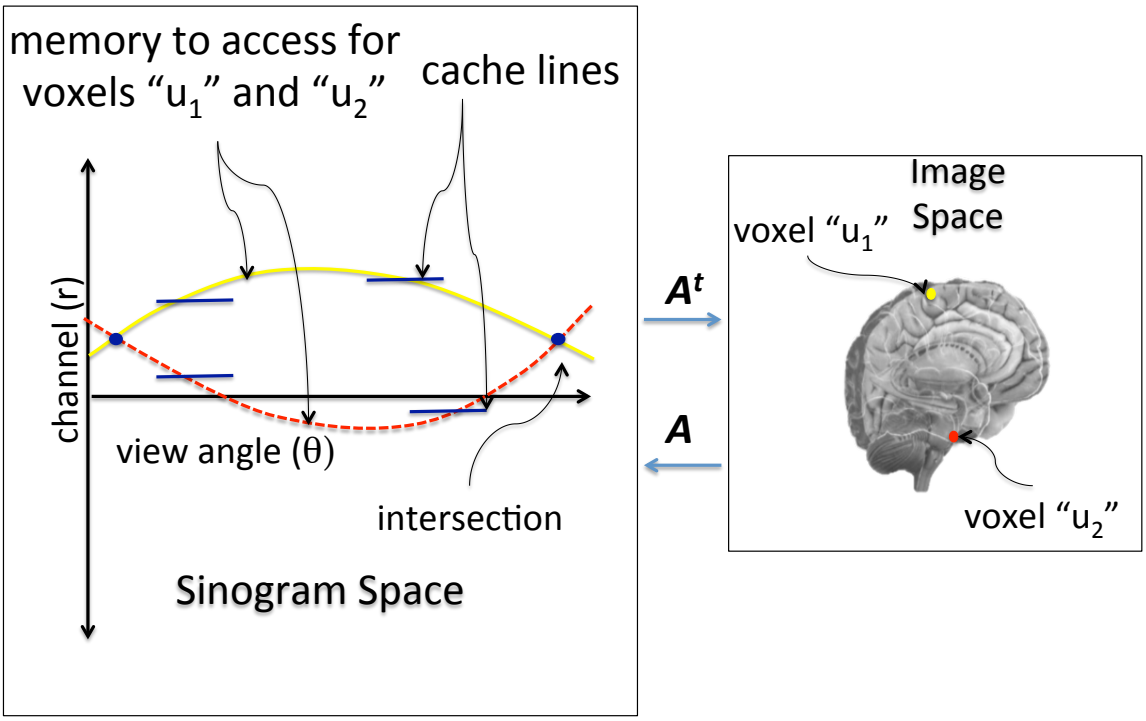


Fig. 2.2.: The sub-figure on the left shows the sinogram space storing all of the CT measurements. The sub-figure on the right shows the final reconstructed image. Notice that each single voxel in the image domain traces out a sinusoidal pattern in the sinogram domain. The sinusoidal pattern for voxel "u_1" is shown in solid yellow while "u_2" is shown in dashed red. In addition, each voxel traces out a different sine wave pattern, but different voxels' sine wave patterns will intersect for some set of views.

for parallelizing ICD, has been demonstrated to only achieve a speedup of $2.3X$ on 16 cores.

2.2 3D COMPUTED TOMOGRAPHY

In this section, we will discuss the CT system setup in which a sample object to be scanned has multiple slices. As discussed before in section 2.1, a CT scanner consists of an X-ray source and a detector array, which are mounted on the opposite ends of a rotating gantry. The object to be imaged is placed near the center of rotation. As the gantry rotates around a fixed z axis, X-rays from the source pass through the object and a series of snap-shot measurements are taken by the detectors at fixed rotation angles, *views* (β). These measurements from the detector elements represent the integral density of the object along the path of the X-rays. To reconstruct a 3D volume of the object, an inverse is computed from the measurements to determine the radio-density of the object. Figure 2.3(a) illustrates the geometry of a CT system, with the X-ray detector elements along the vertical direction called *rows*, and the detector elements along the horizontal direction are channels.

A 3D volume can be obtained by reconstructing the sample object in slices, where a slice is a cross-section of the sample object with a fixed thickness along z axis. In this model, it is important to note that all measurements for a single slice are taken on the same row of the X-ray detector. In Figure 2.3(a), a sample slice within the sample object is shown as a shaded circle and its measurements project to the shaded horizontal line along the same row of the detector array.

Figures 2.3(b), (c) and (d) illustrate how a single voxel maps to the detector measurements. Let u_1 denote a voxel in the sample slice, shown as a red square. Figure 2.3(b) shows the set of channels, represented as a red bar on the detector, that receive measurements from u_1 at view angle 0°; and Figures 2.3(c) and (d) illustrate the measurements from u_1 at view angles 45° and 90°, respectively.

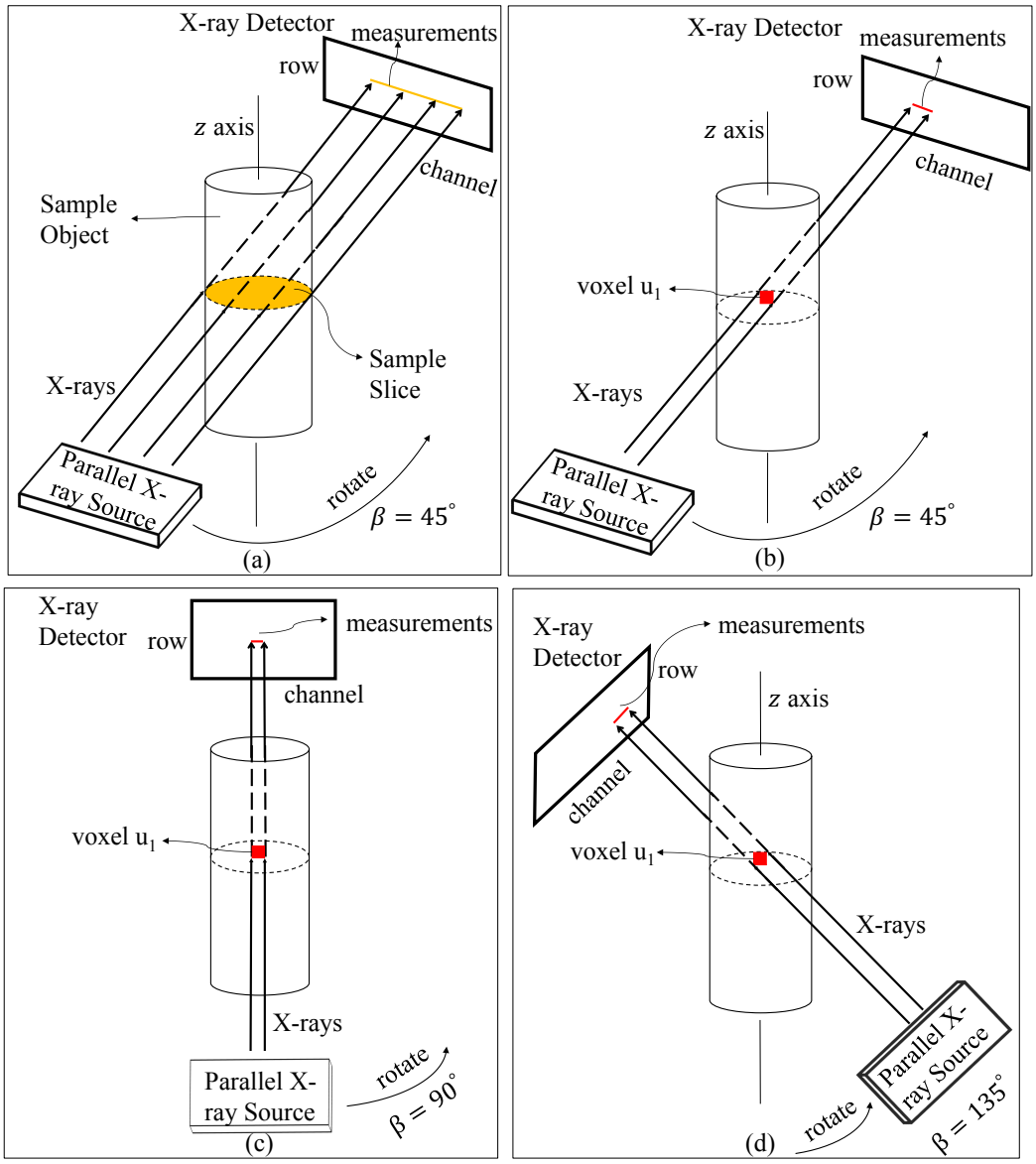


Fig. 2.3.: (a) A Computed Tomography system setup. (b) At view 45°, X-rays pass through voxel u_1, making a projection on the left side of the detector. (c) At view 90°, the projection for voxel u_1 shifts to the center of the detector. Note that the projection at view 90° is shorter than at view 45°. (d) At view 135°, the projection shifts to the right side of the detector.

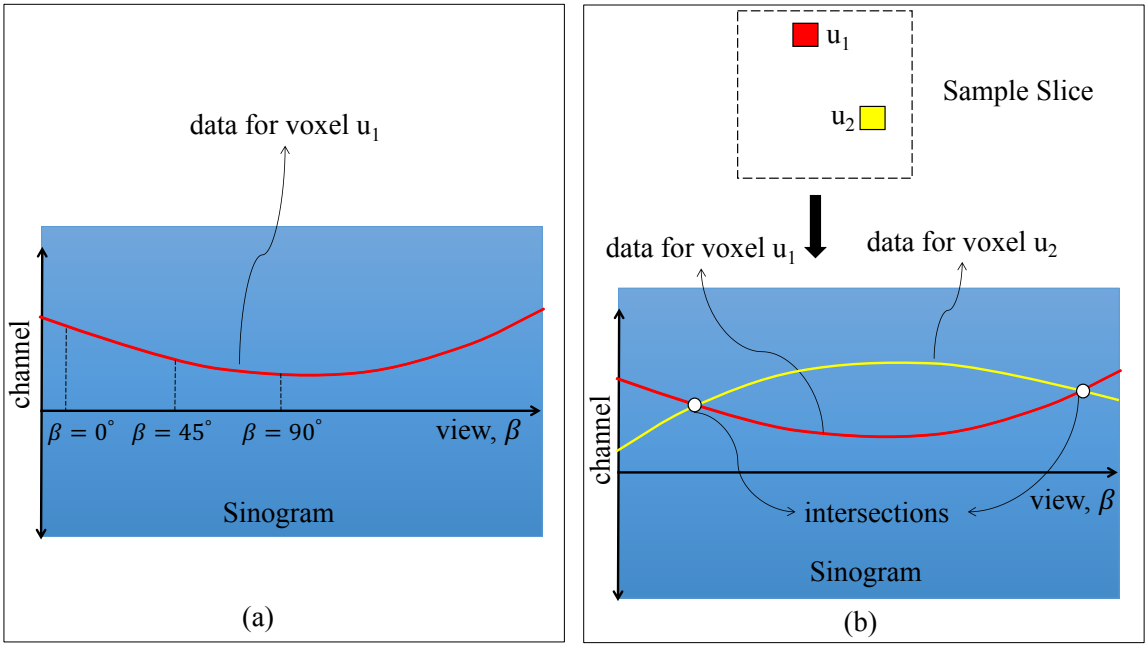


Fig. 2.4.: (a) The update of u_1 follows a sinusoidal trace in sinogram. (b) The sinusoidal traces for u_1 and u_2 intersect with each other at white dots.

Similar to the 2D CT system setup, the measurements for each slice can be organized into sinogram. Figure 2.4(a) illustrates such a sinogram, with the red trace indicating the measurements for the voxel u_1. When u_1 is updated, its associated measurements must be accessed following a sinusoidal path in the computer's memory. Unfortunately, no cache line can access measurements efficiently in memory following a sinusoidal path, which explains why MBIR has poor cache locality. Furthermore as we will see, the amplitude and phase of the sinusoidal path is different for each voxel. So there is no universal reordering of the measurements that can linearize the memory access pattern.

An interesting feature which is not covered in section 2.1 is that for a cubic voxel the width of the voxel trace varies at different view angles. Figures 2.3(b), (c) and (d) illustrate this effect for the projections at the view angles $\beta = 45°$, $90°$, and $135°$. Notice that the projection of the voxel at $\beta = 90°$, shown in Figure 2.3(c), is the shortest; but at angles $\beta = 45°$ and $135°$, shown in Figure 2.3(b) and (d), the projection onto the detector is across the diagonal of the square voxel. So in this case, the width of the trace is the longest and it is increased by a factor of $\sqrt{2}$. Consequently, the combination of varying voxel trace width and sinusoidal memory access leads to irregular data access that inhibits SIMD operations.

Figure 2.4(b) illustrates a key difficulty in performing parallel updates for voxels in the same slice. The update of the two voxels, u_1 and u_2, require access to data along the red and yellow paths in the sinogram, respectively. Notice that the voxel paths for u_1 and u_2 will, in general, intersect at the white dots. Therefore, if u_1 and u_2 are updated in parallel, then synchronizations will be required on the measurements at the intersections of the voxel traces. This observation can be generalized to show that whenever dense view projections are taken, any two or more voxels will overlap at their intersections in the sinogram and synchronizations will be required.

To reduce synchronizations, groups of spatially separated voxels can be updated in parallel, so that their voxel traces are distant from each other [32]. However, such voxel traces also share few measurements, which reduces cache locality. When

groups of spatially close voxels are updated in parallel, their voxel traces are close to each other. Therefore, cache locality improves because voxel updates share many measurements, but the synchronization overhead becomes much worse.

3. MULTI-CORE 2D IMAGE RECONSTRUCTION

3.1 INTRODUCTION

One of the most important research directions in high performance CT image reconstruction can be summarized as addressing two challenges: (1) increasing the amount of parallelism in the reconstruction, and (2) enhancing the locality of data references during the reconstruction. The ICD algorithm presents unique challenges in meeting these two apparently competing goals. Early efforts to speed up ICD focused on parallelizing the algorithm by eliminating dependencies between voxel updates. Parallel methods have been developed to find "loosely coupled voxels" sharing little or no data in common [9,28]. By using loosely coupled voxels, it is shown in [29,30] that, from an algorithmic perspective, updates of voxels could be done simultaneously with little or no negative effect on the convergence rate. Typically, loosely coupled voxels must be far from each other in the same 3D slice, or must be in different slices of the 3D volume. However, while the parallel updates of loosely coupled voxels eliminate the need for shared data, they also hurt cache locality since there is no memory reuse between any two distinct voxel updates. Alternatively, one might choose to update in parallel nearby voxels that are strongly coupled and share more data, thus leading to better cache locality. However, strongly coupled voxels will also have increased data sharing and lock contention across cores, negatively affecting parallelism. Therefore, a good solution must enable good cache locality while also enabling good parallel execution.

To address the competing challenges discussed above, this chapter focuses on 2D single-slice image reconstruction. By constraining the voxel group to be updated in a single slice, the voxel group to be updated is more strongly coupled than voxel groups across different slices. In addition, a slice has enough voxels so that choosing

a loosely-coupled voxel group from them is possible. Therefore, single-slice image reconstruction is a good choice that provides the opportunity to update voxels that are both strongly-coupled and loosely-coupled. Although the 2D single-slice image reconstruction is a simplified case of 3D image reconstruction, the techniques discussed in this chapter, in practice, can be directly transferred to 3D image reconstruction (see Chapter 4) because a 2D reconstruction has the same core computation challenges as a 3D image reconstruction.

In summary, this chapter presents the following contributions:

1. We describe the performance issues inherent in MBIR.
2. We propose the ideas of the *super-voxel* (SV) and *super-voxel buffer* (SVB) to increase locality and prefetching.
3. We identify and discuss available parallelism in MBIR.
4. We propose a parallel super-voxel ICD algorithm (PSV-ICD) that simultaneously addresses the challenges of parallelism and locality.
5. We show experimental results that PSV-ICD leads to an average $14X$ speedup on a single core and a total of $187X$ speedup on 2 Intel Xeon E5 processors with 20 cores in total.

The remainder of the chapter is organized as follows. In Section 3.2, we review background information about the baseline MBIR algorithm. In Section 3.3, we introduce the SV algorithm (SV-ICD/SVB) and its exploitation of cache locality and hardware prefetching. In Section 3.4, we investigate different levels of parallelism in high performance imaging, and introduce the PSV-ICD algorithm. In Section 3.5, we provide an evaluation of SV-ICD/SVB and PSV-ICD using datasets from an Imatron C300 CT Scanner, described in [14], in which we show the achieved performance gains and where they come from. We believe that other domains will benefit from these techniques, and in Section 5.1, we present a discussion of the broader application of these techniques to other problems. Finally, we present our conclusions in Section 3.6.

3.2 THE CONVENTIONAL ICD ALGORITHM

In this section, we will review the central mathematical concepts for MBIR. Then, we will briefly review how the ICD algorithm computes the cost function for MBIR.

MBIR is based on the numerical solution of an optimization problem described by

$$\hat{x} = \underset{x \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(y - Ax)^T D(y - Ax) + S(x) \right\} \tag{3.1}$$

We consider the image x as a N dimension vector whose elements are voxels and the data y as a vector of size M containing the CT measurement data. To relate this information with Figure 2.2, M is the number of channels times the number of views and N is the number of voxels in the image space. The matrix A is a $M \times N$ forward system matrix of the scanner geometry, D is a $M \times M$ diagonal weighing matrix containing the inverse variance of the scanner noise and $S(x)$ is the regularizing prior function which depends upon voxels only. The system matrix, A, encodes the geometry of the scanner, and each element $A_{i,j}$ roughly corresponds to the length of intersection between the j^{th} voxel and the i^{th} projection. Since a single projection only intersects a small subset of the voxels, the matrix A is very sparse and is typically either precomputed and stored as a sparse matrix or computed on the fly. The diagonal matrix D is also pre-computed from the data so that each element, $D_{i,i}$, corresponds to the inverse variance of the measurement y_i. In the rest of this chapter, diagonal entry $D_{i,i}$ is abbreviated as d_i. The ICD algorithm for solving Equation (3.1) works by updating each voxel in sequence to minimize the overall cost function while keeping the remaining voxels fixed. Formally, the update of the j^{th} voxel, x_j, is given by

$$\hat{x}_j = \underset{x_j \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(y - Ax)^T D(y - Ax) + S(x) \right\} \tag{3.2}$$

If we denote $\hat{x}_j$ to be the j^{th} voxel after the update and $\hat{x}_j = x_j + \alpha$, where α is the change of the update, then Equation (3.2) can then be simplified as

$$\hat{x}_j = \arg\min_{\alpha \geq -x_j} \left\{ \frac{1}{2}(y - A(x + \alpha\epsilon_j))^T D(y - A(x + \alpha\epsilon_j)) + S(x + \alpha\epsilon_j) \right\} \tag{3.3}$$

where ϵ_j is a vector whose j^{th} element is 1, and 0 otherwise. In practice, the computation associated with the term $S(x+\alpha\epsilon_j)$ is negligible in the ICD update because it only includes a small number of local operations. So we will focus on the remaining quadratic term of Equation (3.3).

Direct implementation of the ICD algorithm using Equation (3.3) requires the evaluation of the term $y - Ax$ with each voxel update. Fortunately, there is a simple method for speeding the computation by keeping a state variable to store this error term $e = y - Ax$. Therefore, Equation (3.3) can then be further simplified as:

$$\hat{x}_j = \arg\min_{\alpha \geq -x_j} \left\{ \frac{1}{2}(e - A\alpha\epsilon_j)^T D(e - A\alpha\epsilon_j) + S(x + \alpha\epsilon_j) \right\} \tag{3.4}$$

Since the first term in Equation (3.4), $\frac{1}{2}(e - A\alpha\epsilon_j)^T D(e - A\alpha\epsilon_j)$, has a quadratic form, the first term of Equation (3.4) can be expressed as a simple quadratic function:

$$\hat{x}_j = \arg\min_{\alpha \geq -x_j} \left\{ \frac{1}{2}(e^T De - 2e^T DA\epsilon_j\alpha + \epsilon_j^T A^T DA\epsilon_j\alpha^2) + S(x + \alpha\epsilon_j) \right\} \tag{3.5}$$

In Equation (3.5), the constant term, $\mathrm{e}^T De$, is not relevant to the cost function minimization and this constant term is dropped. To minimize Equation (3.5) and update the voxel x_j, it is necessary to compute the first two derivatives of the cost function, namely $-e^T DA\epsilon_j$ and $\epsilon_j^T A^T DA\epsilon_j$. These two derivatives, denoted by θ_1 and θ_2, are given by

$$\theta_1 = -e^T DA\epsilon_j = -e^T DA_{*,j} = -\sum_{i=1}^{M} d_i A_{ij} e_i \tag{3.6}$$

$$\theta_2 = \epsilon_j^T A^T DA\epsilon_j = A_{*,j}^T DA_{*,j} = \sum_{i=1}^{M} d_i A_{ij}^2 \tag{3.7}$$

where $A_{*,j}$ is the j^{th} column of the matrix A and e_i is the i^{th} element in the error term.

By replacing $-e^T DA\epsilon_j$ and $\epsilon_j^T A^T DA\epsilon_j$ with notations θ_1 and θ_2, one can then write the minimization of the 1-D objective function for x_j explicitly as

$$\hat{x}_j \leftarrow \underset{\hat{x}_j \geq 0}{\operatorname{argmin}} \left\{ \theta_1 \hat{x}_j + \frac{\theta_2(\hat{x}_j - x_j)^2}{2} + f(\hat{x}_j, x_{\partial j}) \right\}, \tag{3.8}$$

where x_j is the j^{th} voxel's value before the update, and $(\hat{x}_j, x_{\partial j})$ is a prior function of the eight neighbors of the voxel x_j. In addition, we sometimes skip the update of voxels in a process called "zero-skipping". Zero-skipping can be used to speed up convergence and it is typically performed whenever a voxel and all its 8 neighbors are zero [28].

Algorithm 1 Voxel Update (j, x, e, d)

1: $x_j^{'} \leftarrow x_j$
2: $\theta_1 = \sum_{i=1}^{M} d_i A_{ij} e_i$
3: $\theta_2 = \sum_{i=1}^{M} d_i A_{ij}^2$
4: $\hat{x}_j \leftarrow \text{argmin}_{\hat{x}_j \geq 0} \left\{ \theta_1 \hat{x}_j + \frac{\theta_2 (\hat{x}_j - x_j)^2}{2} + f(\hat{x}_j, x_{\partial j}) \right\}$
5: $e \leftarrow e + A_{*j}(\hat{x}_j - x_j^{'})$

The pseudo code for the voxel update algorithm is shown in Algorithm 1 published in [28]. The first step is to copy the voxel value to a local register $x_j^{'}$. We compute θ_1 and θ_2 using the formulas in Equation (3.6) and (3.7). Then we compute the voxel's updated value $\hat{x}_j$ efficiently by solving a 1-D optimization problem [28] in the Equation (3.8). Finally, we update the error term by forward projecting the update step $\hat{x}_j - x_j^{'}$. When implemented properly [28], the computation of Algorithm 1 is dominated by steps 2, 3, and 5. The remaining steps typically represent a negligible amount of additional computation. In each iteration of the conventional ICD algorithm, a voxel is updated exactly once. However, the order of voxel updates may vary across different iterations. For CT image reconstruction, experimental results have indicated that selecting voxels in random order provides significantly faster convergence than raster order, as the correlation between successive updates is reduced [33]. Therefore, the random update order is typically used in the conventional ICD algorithm. The ICD algorithm further speeds up convergence by focusing computation on regions of the image that contain fine detail, such as edges [34]. When a voxel is chosen for update, the reconstruction edge mask is first checked and updates are calculated only for the masked voxels.

3.3 IMPROVING SEQUENTIAL EXECUTION

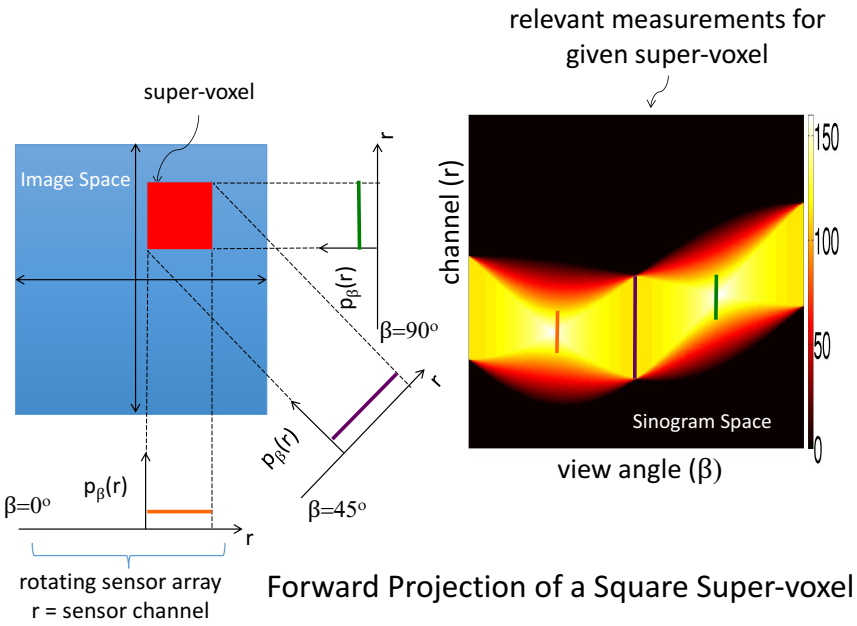
Good single core performance is essential for overall good parallel algorithm performance. In this section we discuss the concept of the super-voxel (SV) and how SVs and the super-voxel buffer (SVB) can improve the locality and single-core performance of the image reconstruction algorithm.

3.3.1 SUPER-VOXELS

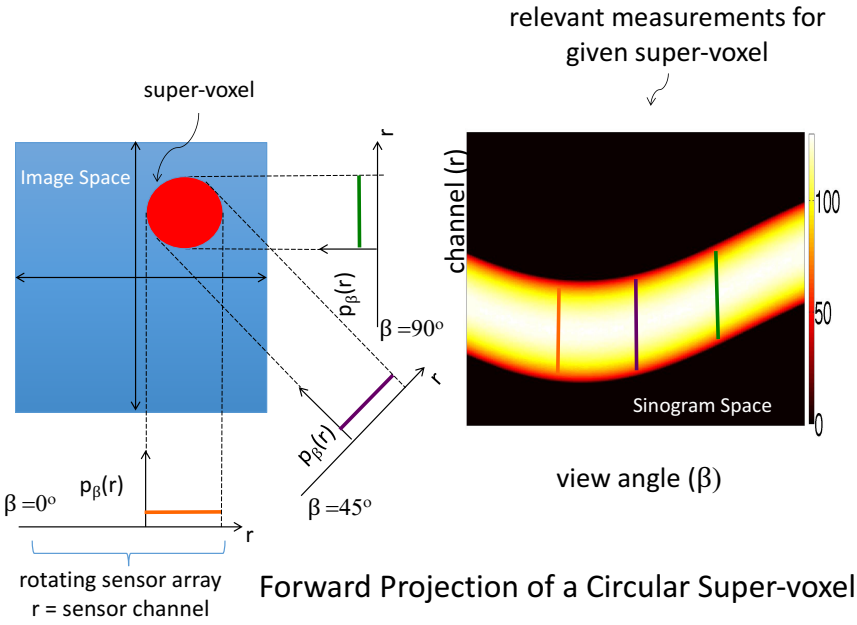
A SV is a group of voxels that are contiguous in the image space and whose measurement data (from scanners) is likely to be close together in sinogram memory. Thus, operating on the data for these voxels is likely to increase temporal and spatial locality.

An initial question is what voxels should be in a SV to increase cache locality? To answer this question, the *forward projection* [35] of a set of voxels can be used to determine where in the sinogram space the corresponding data values lie. For example Figure 3.1(a) shows the forward projection for a square SV. At each view angle β, we record the SV's projection on a rotating sensor array. The more sensor channels that detect a projection at that view angle, the more measurement data will be in that view in the sinogram space. In addition, brighter points (more yellow and white) in the sinogram space indicate higher levels of data reuse in the SV. From Figure 3.1(b), we can see that a circular SV is the ideal choice for the SV shape because it will have the least amount of measurement data and the highest data reuse[1]. Circular SVs, however, do not tessellate the image space. Consequently, circular SVs must have a large amount of overlap to fully cover the image space, leading to excessive computations on the voxels in the multiple overlapping SVs. At the same time, SVs tessellating the image space with no overlap will also lead to excessive computation because the algorithmic convergence could be adversely affected at SV boundaries.

[1]This result holds if the sinogram is formed by a dense sampling of views at all angles



(a)



(b)

Fig. 3.1.: (a) The forward projection of a square SV. (b) The forward projection of a circular SV. Notice that the circular SV maps to a band of uniform thickness in the sinogram. The brighter color in the sinogram also corresponds to more data reuse.

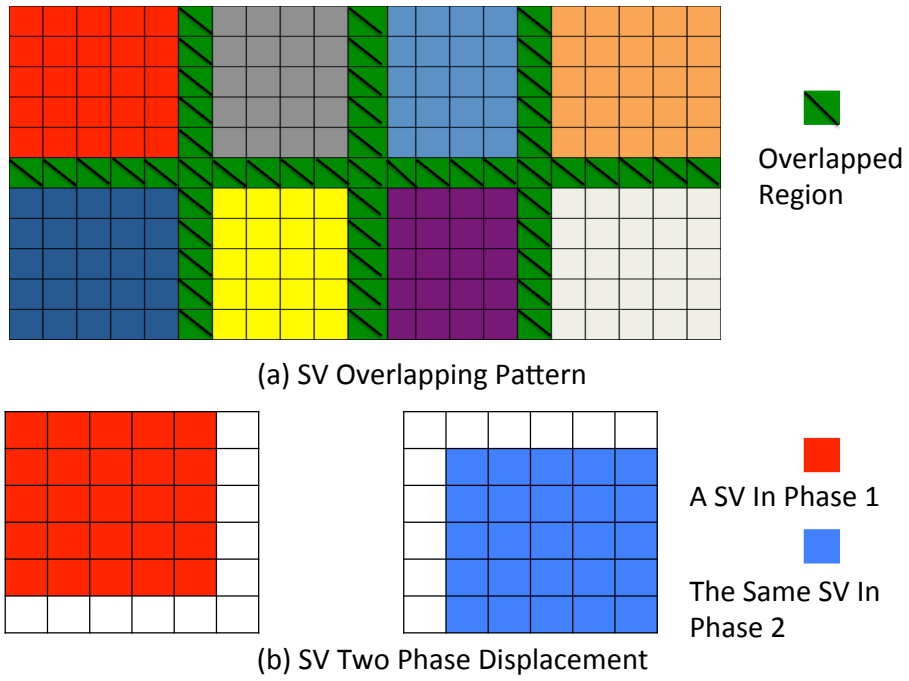


Fig. 3.2.: The layout and the overlaps of SVs in the image space using the two-phase mechanism. In (a), shaded green cells represent overlapping neighboring SVs. In (b), the red SV is a phase 1 SV and the blue SV is the same SV in phase 2 but with a diagonal displacement.

To overcome these flaws, we use square SVs and a two phase convergence mechanism. When we use square SVs, the data reuse frequency is lower compared to the circular SVs, as can be seen in the forward projection for the square SV in Figure 3.1(a). However, having fewer voxels in the overlapping regions and eliminating redundant computations more than compensate for this disadvantage. Figure 3.2 illustrates the SV layout used to achieve the two phase convergence. In Figure 3.2(a), each phase of SVs of side length 6 provides a full coverage of the image space while adjacent SVs overlap at the boundaries. This introduces a small amount of redundant computations but it is compensated for by having faster algorithmic convergence. In Figure 3.2(b), the image on the left shows a SV in Phase 1 and the image on the right shows the same SV in Phase 2 but with a diagonal displacement of 1. All SVs in the odd iterations of the algorithm use Phase 1 while SVs in the even iterations use Phase 2. Both overlapping adjacent SVs and fixed distance phase change ensure more rapid algorithmic convergence at SV boundaries. The detailed convergence analysis is shown in Figure 3.8 of the Experimental Results section.

3.3.2 SUPER-VOXEL BUFFERS

As discussed above, SVs improve performance in part by reducing data cache misses. Additional improvements can be achieved by improving the ability of the hardware to perform prefetching. Hardware prefetching (or just prefetching) is performed by modern processors when they can recognize one or more sequences of accessed data, typically with accesses of the form $i \pm k$ for a linear sequence of is. When such a sequence is recognized, the hardware predicts memory locations that are likely to be accessed soon and prefetches them into cache before they are needed. SVs do not improve the prefetch rate because most SVs are not centered in the image and therefore the corresponding measurement data in the SV is in a sinusoidal band with access patterns that are not recognized by the prefetcher. Figure 3.3 graphically illustrates this band pattern in which a red block represents the SV to be updated in

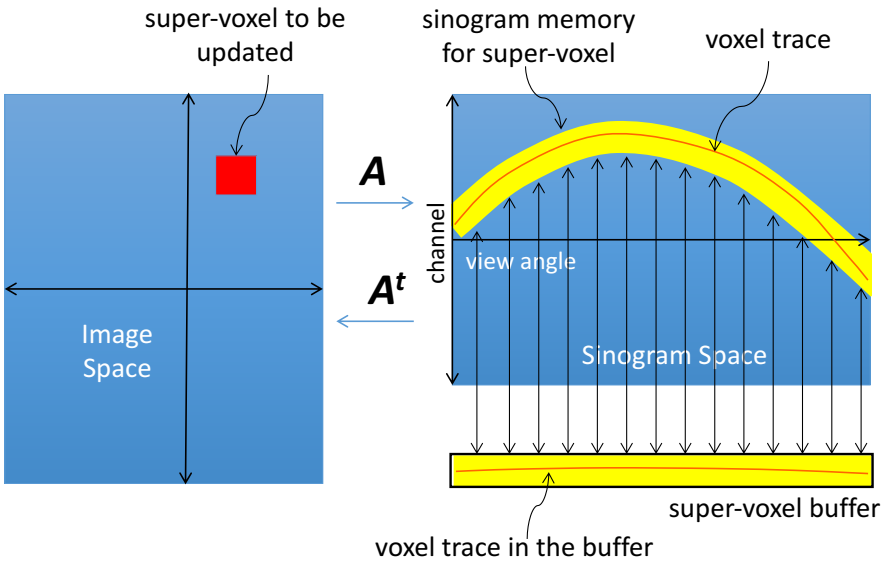


Fig. 3.3.: Memory access in sinogram space for SV updates. Processor hardware is used to load sinogram memory into a SVB. Note that a voxel trace is straightened out in the SVB.

the image space and sinogram entries related to this block are illustrated as a yellow band in the sinogram space.

In creating a super-voxel buffer (SVB), the memory accesses in the yellow band in Figure 3.3 are copied into the SVB shown in the same figure. The SVB lays out the SV data so that memory accesses follow an almost "straight line" pattern that is good both for hardware prefetching and ensuring that a prefetched cache line contains entirely, or almost entirely, useful data. The sinogram band can be approximated as a band sine function while the SVB is a straight band. Therefore the transformation

from the sinogram band to SVB can be described as the inverse function of the sinogram band's sine function.

Algorithm 2 SV-ICD/SVB (G, e, d)

INPUT: G: a CT image consisting of voxels. d: diagonal entries of weighing matrix.

OUTPUT: e: error term

LOCAL: SVB_e, SVB_d SVBs to store local memory accesses

1: Initialize e and create a tessellating SVs set P
2: **for** super-voxel $a \in P$ **do**
3: SVB_e,$SVB_d \leftarrow \emptyset$
4: **for** each view β in the sinogram **do**
5: $SVB_e \leftarrow Copy\ e$ from sinogram to SVB
6: $SVB_d \leftarrow Copy\ d$ from sinogram to SVB
7: **end for**
8: **for** super-voxel $a \in P$ **do**
9: **for** Voxel $x_j \in a$ **do**
10: Voxel Update(j, x, SVB_e, SVB_d)
11: **end for**
12: **end for**
13: **for** each view β in the sinogram **do**
14: $e \leftarrow Copy\ SVB_e$ from SVB to sinogram
15: **end for**
16: **end for**
17: **if** image not converged **then**
18: SVs set P shifts phase. Go back to step 2.
19: **end if**

The detailed SV and SVB implementation is summarized in Algorithm 2 (SV-ICD/SVB). We first tessellate the image space by using the two phase convergence mechanism, and then update voxels sequentially within each SV. In addition, we use

two SVBs, called SVB_e and SVB_d, to store e (the error term) and d (the diagonal entries of the weight matrix), respectively, because we need to access both of them in the ICD algorithm. At the end of the Algorithm 2, revised memory data are copied back from the SVB to e. No data are copied from an SVB to d because it is not modified in Algorithm 1.

3.3.3 THEORETICAL ANALYSIS FOR SUPER-VOXEL BUFFERS

In this section, we will provide a theoretical analysis on the average number of consecutive memory locations accessed in the SVB for a single voxel update along the view direction, denoted Run_{SVB}. To illustrate how Run_{SVB} is computed, we will use Figure 3.4 for this purpose. In Figure 3.4, r_j is a voxel trace in the SVB whose average absolute slope is $m(\phi)$ (computed in Appendix D). In addition, we define $\phi = \arctan(m(\phi))$. Then, we find a point B on the voxel trace r_j, such that the distance from B to view (β) axis (illustrated as line segment BC in Figure 3.4) equals to the detector channel spacing, Δ_d. In addition, we find a point A on the intersection between the voxel trace r_j and view (β) axis and we make ϕ to be the angle between BA and AC line segments. Notice that r_j is almost flattened such that a small segment of r_j can be approximated as a straight line. Therefore, $|\tan\phi|$ equals to the ratio of BC line segment length to AC line segment length. We can then express $|\tan\phi| = \frac{\Delta_d}{R}$, where R is defined as the distance from point A to point C. In summary,

$$|\tan\phi| = m(\phi) = \frac{\Delta_d}{R}, \tag{3.9}$$

In addition, we can compute R as such:

$$R = \Delta_\beta \cdot Run_{SVB}, \tag{3.10}$$

where Δ_β is the view angle spacing in radian and it equals to $\frac{\pi}{N_v}$. Combining Equation (3.9) and Equation (3.10), we can then compute Run_{SVB}, given by

$$Run_{SVB} \approx \frac{3N_v\Delta_d}{N_{wh}\Delta_x(1+\sqrt{2}+\ln(1+\sqrt{2}))} \tag{3.11}$$

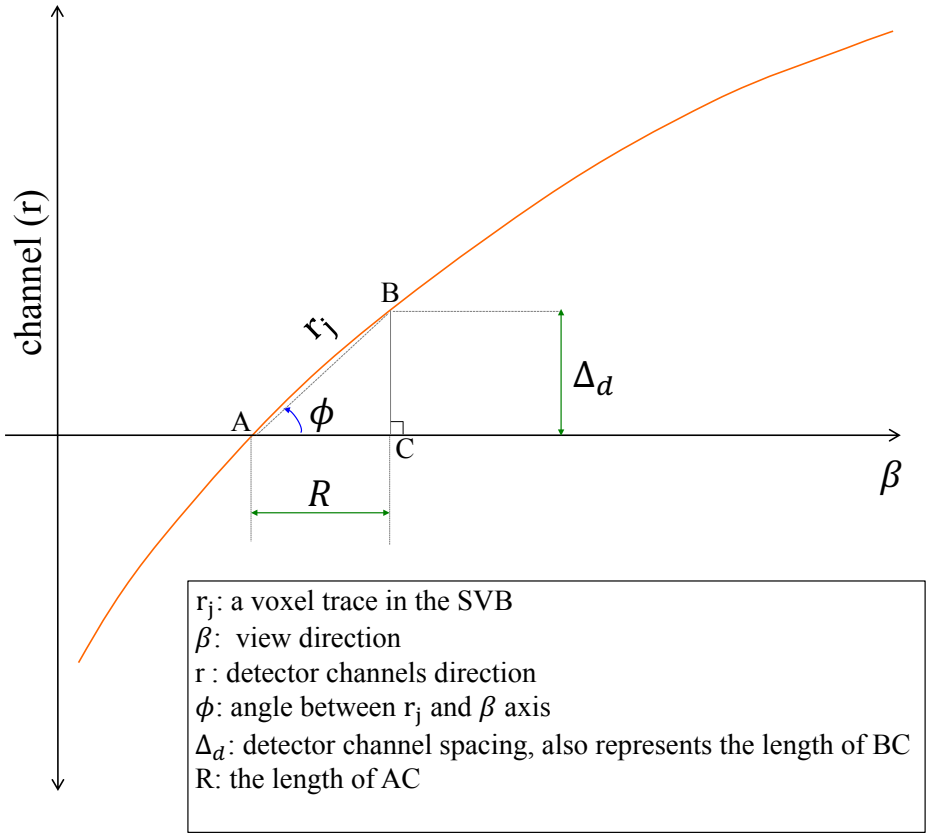


Fig. 3.4.: Shows how Run_{SVB} is computed. In this figure, angle ϕ equals to the average absolute slope for the voxel trace r_j.

where N_v is the number of views, N_{wh} is the number of voxels on each side of the square SV and Δ_x is the voxel width. For the example where $N_{wh} = 13$, $N_v = 720$, and $\Delta_d = \Delta_x$, it can be shown analytically that the average run length of data is 50. Alternatively, if the entire image is one SV (i.e., the SV concept is not used), then $Run_{SVB} \approx 1$ so that on average only a single value of data can be read in sequence in the SVB for a voxel update. A more thorough analysis about the impact of SVBs on performance is discussed in Section 3.5, showing an average tenfold increase of the prefetch hit rate.

3.4 IMPROVING PARALLEL EXECUTION

In this section, we first discuss different levels of possible parallelism in high performance CT image reconstruction.

Hierarchical Parallelism Achieving a high degree of parallelization is extremely important for high performance CT reconstruction. For any number of CPU cores used in the image reconstruction, sufficient parallelism is needed to efficiently use the cores. For future work with GPUs and large clusters, thousands of cores will need to be kept busy. We have identified the 4 major levels of parallelism listed below.

1. Intra-voxel parallelism: parallelism within the update of a single voxel.
2. Intra-SV parallelism: parallelism across the updates of multiple voxels in a single SV.
3. Inter-SV parallelism: parallelism across multiple SVs in a single image.
4. Inter-slice parallelism: parallelism of multiple slices (images) of a single 3D reconstruction.

Among these 4 levels of parallelism, inter-slice parallelism is widely used by others, e.g. [9, 18, 28, 36]. In a 400 slice reconstruction, at least 200 slices can be processed in parallel with little or no communication required among processes. The data vectorization in computing Equation (3.6) and (3.7) in Section 3.2 can be considered to be an example of intra-voxel parallelism since it parallelizes multiple data operations in a single voxel update. Moreover, GCD [30] can be taken as an exemplary intra-SV parallelism where the entire image is considered to be a single SV.

Inter-SV parallelism (with each SV containing more than one voxel) has fewer conflicts on shared data than intra-SV parallelism, reduced communication overhead and sufficient work to keep every core busy. Intuitively, fewer conflicts mean that inter-SV parallelism allows each core to have more work to do before a possible multi-core communication. In addition, the frequency of communication is also reduced proportionally to the SV size.

Moreover, these four major levels of parallelism are largely orthogonal, so that the total number of parallel operations increases as the product of the number of parallel threads in each level. This is important because the number of cores in modern computing systems is rapidly increasing with each new generation of device. Hence, keeping all these cores efficiently busy is a crucial requirement for fast efficient MBIR. In addition, it is also important to note that these four levels of parallelisms are hierarchical since one slice is a set of SVs and each SV is a set of voxels.

Algorithm 3 GCD ($voxelSet$, e)

1: **for** each voxel x_j in the $voxelSet$ **do in parallel**
2: $\tilde{x_j} \leftarrow x_j$
3: $\theta_1 \leftarrow Compute$ as in Equation (3.6)
4: $\theta_2 \leftarrow Compute$ as in Equation (3.7)
5: $x_j \leftarrow \text{argmin}_{r \geq 0}\{\theta_1 r + \frac{\theta_2 (r - \tilde{x_j})^2}{2} + f(r, x_{\partial j})\}$
6: lock
7: $e \leftarrow e + A_{*j}(x_j - \tilde{x_j})$
8: unlock
9: **end for**
10: **if** image not converged **then**
11: Go back to step 1.
12: **end if**

The challenges of parallelism The fundamental challenge in a parallel MBIR is the trade-off between cache locality and parallelism discussed in Section 3.1. Because of the sinusoidal nature of voxel traces in the sinogram space, it can be shown analytically that any two traces are guaranteed to have at least one intersection, as illustrated in Figure 2.2. We call those voxels whose traces have many intersections "strongly coupled voxels" and those voxels whose traces have few intersections "loosely coupled voxels".

Strongly coupled voxels have more shared measurement data in the sinogram and take advantage of cache locality, but the algorithmic convergence will be slower. Algorithm 3 (the GCD algorithm) uses one kind of intra-SV parallelism and violates dependencies from the update of e in step 7 to its use in step 3 of the algorithm. This violation of dependencies leads to a correct answer because of the convergent nature of the algorithm, but it will converge much more slowly, i.e., it will require more voxel updates and computations. In addition, a more fundamental limitation comes from the significant waiting time for a lock in GCD. As we can see from Algorithm 3, every voxel update has a lock waiting overhead. This lock contention is a source of significant overhead in parallel executions.

On the other hand, loosely coupled voxels have less shared measurement data in the sinogram and thus cache locality suffers. However, the algorithmic convergence will be faster because the voxel traces have fewer intersections. In the case of the GCD algorithm, a common solution to this problem is to iterate over the voxels such that widely separated voxels are simultaneously updated (see [30]) and therefore there are fewer intersections in the voxel traces. Although this will also lead to better parallel performance because the lock waiting overhead is reduced, the cache locality is reduced because there is little sharing of measurement data.

Our solution In this section, we describe our parallel SV ICD algorithm, called PSV-ICD. It uses inter-SV parallelism and maintains both good cache locality and good parallel performance. To speed up convergence, we operate on multiple SVs that are far apart in the image space, similar to what GCD does with voxels. The distance between these SVs also minimizes interference and lock contention resulting from simultaneous SVs update. Each SV has its own SVB, and within a SV updates are performed sequentially, as in Section 3.3, ensuring good cache locality. We call the technique with multiple SVs and SVBs *Augmented SVB*. Algorithm 4 shows the detailed implementation of this parallel SV algorithm (PSV-ICD). In the beginning of the program, each computing core is assigned with one SV in step 4, and then each

Algorithm 4 PSV-ICD (G, e, d)

INPUT: G, d as before in Algorithm 2

OUTPUT: e: error term

LOCAL: SVB_e^k, SVB_d^k as before in Algorithm 2

LOCAL: $(SVB_e^k)'$ a temporary copy of SVB_e^k

LOCAL: $SVB_{\Delta e}^k$ error buffer for sv^k

1: Initialize e and create a tessellating SVs set P
2: **while** P is not empty **do**
3: Identify $sv_{subset} \subset P$
4: **for** super-voxel $sv^k \in sv_{subset}$ **do in parallel**
5: Create SVB_e^k, SVB_d^k as in Algorithm 2
6: $(SVB_e^k)' \leftarrow SVB_e^k$
7: update all voxels $x_j \in sv^k$ as in Algorithm 2
8: $SVB_{\Delta e}^k \leftarrow SVB_e^k - (SVB_e^k)'$
9: Lock
10: $e \leftarrow e + SVB_{\Delta e}^k$
11: Unlock
12: **end for**
13: $P \leftarrow P \setminus sv_{subset}$
14: **end while**
15: **if** image not converged **then**
16: SVs set P shifts phase. Go back to step 2.
17: **end if**

core creates its own SVB, denoted SVB^k for the k^{th} SV, by copying needed memory accesses into private, local per-core caches in step 5.

The augmented SVB, illustrated in Figure 3.5, enables inter-SV parallelism. Because multiple SVs are being operated on simultaneously, pressure on shared caches is increased. Tuning the size of the SVs can control the pressure, and the multiple SVBs allow each core to access its data without interfering with other cores. A direct benefit of this is reduced contention on shared memory locations that are being updated. Another benefit is a reduction in both false and true sharing on caches. As shown in Figure 3.7 (c), this algorithm, on average, leads to a significant $187X$ speedup on 20 cores over the baseline ICD. The algorithm also uses the same two phase design described in Section 3.3.1, resulting in convergences as fast as the sequential algorithm, as seen in Figure 3.8 of the Experimental Results section.

The challenge with inter-SV parallelism is that the SVBs for different SVs will overlap in the sinogram domain, as shown in Figure 3.5. This means the individual SVBs' data must be combined into the full sinogram at the end of SV updates. At the end of SV updates, each SVB^k contains updated sinogram data for the $k^{th}SV$ only. A simple solution, such as the lock and unlock used in Algorithm 3, will not work for simultaneous SV updates since if a SV updates the full sinogram without taking into account updates by other SVs, the other SVs' updates will be overwritten and lost. In order to solve this problem, we create a second SVB error buffer for each SV. This error buffer, denoted $SVB^k_{\Delta e}$, is initialized to 0 and all changes of data relative to the k^{th} SV are made to this buffer only. When SV k has been updated, the sinogram data changes kept in $SVB^k_{\Delta e}$ are atomically added to the full sinogram, updating and not overwriting the result. In addition to combining individual SVBs' data into the full sinogram, this approach can also significantly reduce lock waiting time. Intrinsically, this approach reduces all of the local updates to one update to the full sinogram. By using this approach, voxel updates within a SV can be performed asynchronously while other SVs are being processed, so lock contention within a SV can be eliminated. The lock waiting overhead exists only after all voxels within a SV

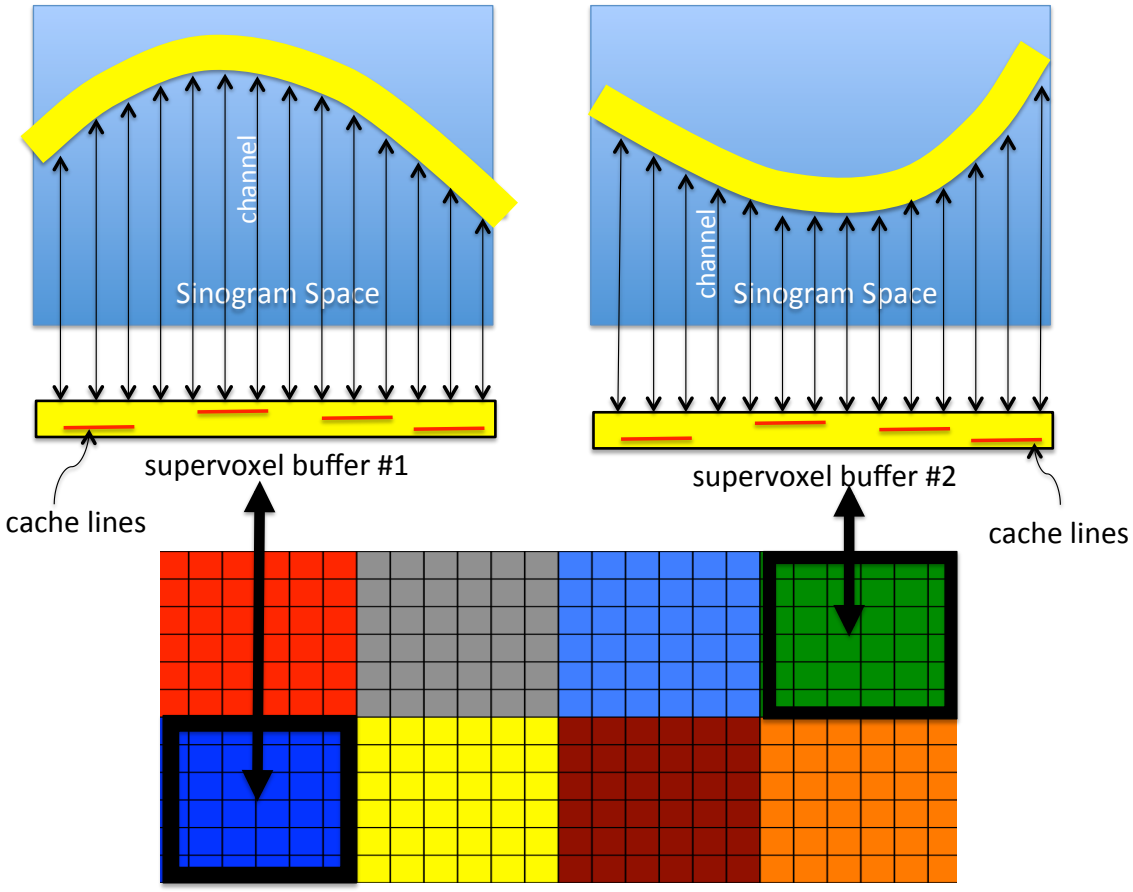


Fig. 3.5.: Parallel SVs update with separate augmented SVBs. Two different SVs have different but overlapping sinogram bands.

are updated. In Algorithm 4, $SVB^k_{\Delta e}$ is created in step 8 after all voxels in the SV are updated and it represents the additions of data made to this buffer. In step 10, these additions of data are atomically added back to the memory.

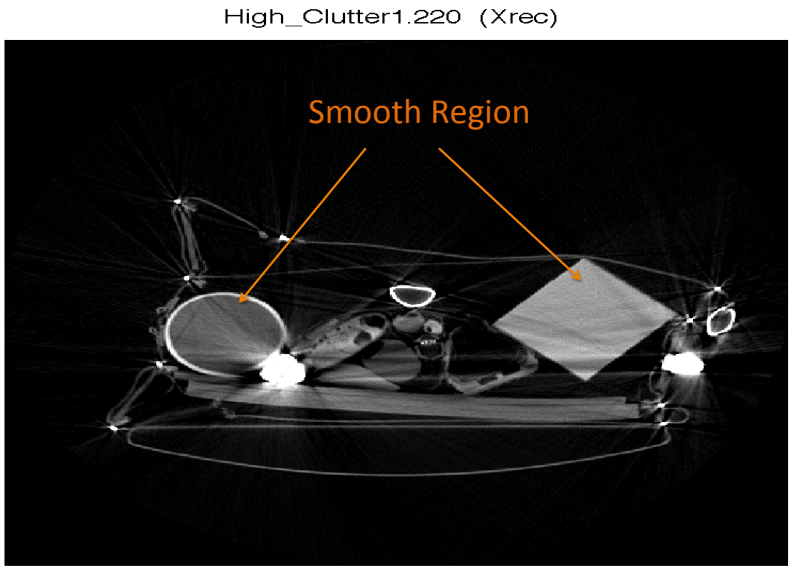
3.5 EXPERIMENTAL RESULTS

In this section, we apply various CT image reconstruction algorithms to a benchmark data set containing 3200 test cases obtained from an Imatron C-300 scanner in the ALERT Task Order 3 (TO3) study [14]. Each slice in this data set has the following specification: (1) parallel beam projection; (2) 720 views uniformly distributed between 0 and 180 degrees; (3) 1024 channels uniformly sampled over the region of interest (ROI); and (4) 512×512 reconstructed image size with an embedded circular ROI.

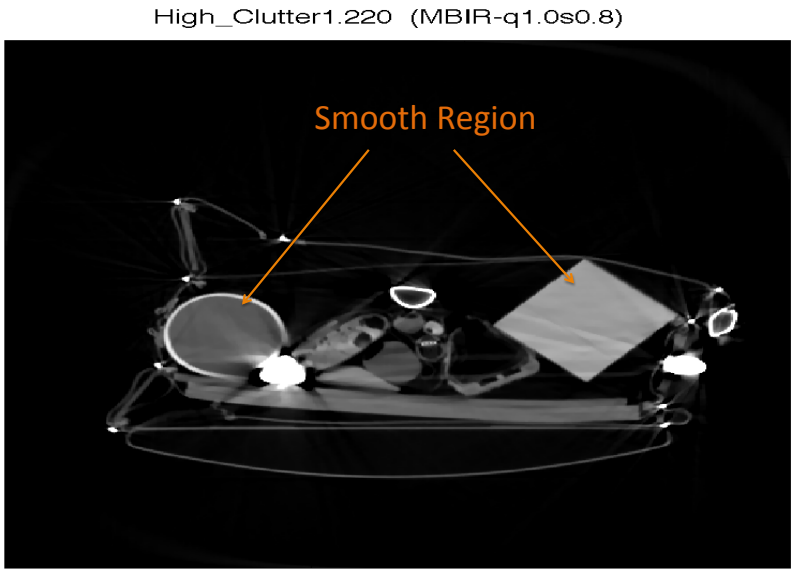
Figure 3.6 is an example slice in the data set reconstructed by both FBP and MBIR. Notice that MBIR has much higher image quality with less noise in the smooth region of this slice. We choose this data set because it has been confirmed as the standard CT data set in ALERT task order efforts and the number of views, channels and selected image resolution for this data set are typical of real image scanners.

To summarize our empirical results, Table 3.1 lists results for all algorithms discussed in this dissertation. Because the baseline ICD, SV-ICD/SVB and PSV-ICD do not update all voxels (see Section 3.2 and [28]) in each iteration, we measure convergence using *equivalent iterations* or *equits*. Each N voxel updates, where N is the number of voxels in the image, is one equit. In addition, we evaluate algorithmic convergence by measuring the root-mean-square error (RMSE) in Hounsfield Units[2] between the result and a fully converged reconstruction after approximately 20 equits. In extensive experimentations in the past, we have found that an RMSE of less than 10 HU consistently results in high quality reconstructions with little or no visible convergence artifacts. Therefore, all reconstructions in this table are converged to reach less than 10 HU of RMSE. All data in this table was collected on a node containing two standard 2.6 GHZ clock rate Intel Processors Xeon-E5 2660 v3 with 10 cores. Each core has an L1 data cache of size 32 KB and an L2 cache of 256 KB. Each core also has a shared L3 cache of 25 MB. All of the algorithms in the experimental results have

[2]The Hounsfield Unit (HU) is a CT measurement unit of the object's radio density comparing with the radio density of distilled water.



(a)



(b)

Fig. 3.6.: One reconstructed slice example in Imatron C-300 scans by FBP and MBIR. The image on the top is this slice fully converged by FBP. The image at the bottom is done by MBIR. Note the image quality enhancement in the smooth region.

Table 3.1.: Table of performance measurements for all algorithms discussed in this dissertation. Column 2 is the baseline ICD code. Column 3 is SV-ICD/SVB. Columns 4-6 are PSV-ICD with 4 cores, 16 cores and 20 cores respectively. Note that row 6, T_r, is the average actual reconstruction time and row 8 is the speedup of T_r at 20 cores comparing with the sequential baseline ICD.

Factor	Baseline ICD	SV-ICD/SVB	PSV-ICD(4)	PSV-ICD(16)	PSV-ICD(20)
O_F (GFLOP)	83	83	332	1331	1664
N_F (GFLOP)	18.52	18.38	18.38	18.38	18.38
N_e (equits)	4.6	4.0	4.1	4.1	4.2
E_F (efficiency)	0.41%	5.95%	4.0%	3.34%	3.70%
T_r (sec)	253	15	5.6	1.8	1.35
T_{opt} (sec)	1.03	0.89	0.22	0.06	0.05
Speedup		$16.86X$	$45.17X$	$140.55X$	$187.40X$

been implemented in OpenMP using the Intel C++ Compiler version 15.0.3.187 and SIMD pragmas to specify parallelization and data vectorization. In addition, optimization level O3 was used in compiling codes. The baseline ICD refers to the conventional ICD algorithm discussed in Section 3.2 with 1 core. The baseline code is accessible at https://engineering.purdue.edu/ bouman/software/tomography/mbirct/ [37] and is used by researchers in many real-world applications, for example, [9, 10, 15]. The SV reconstruction in this table uses square SVs of side length 13 because it is empirically shown to be the best choice for the SV size for this processor.

To better understand and quantify our results, we use a very simple model of computation time given below. The computation time given below is the average computation time per slice. As described in Section 3.4, inter-slice parallelism has been widely used with good parallel performance [9]. Our experiments focus on average computation time per slice and the parallel performance within each slice, since this is considered to be the most difficult case. We compute the average reconstruction time as:

$$T_r = \frac{N_F N_e}{O_F E_F} \tag{3.12}$$

where:

1. T_r = average actual reconstruction time.
2. N_F = average number of single precision billion floating point operations (GFLOP) per equit.
3. N_e = average number of equits required for reconstruction.
4. O_F = average theoretical single precision GFLOP per second of CPU.
5. E_F = average GFLOP efficiency. E_F is the ratio of T_{opt} to T_r where T_{opt} is the theoretical execution time in seconds given 100% GFLOP efficiency.

Using this framework, Table 3.1 quantifies the result of our experiments on 3200 slices in the TO3 data set. Notice that we achieved over 187 speedup, on average, over the single core baseline algorithm by using the PSV-ICD algorithm on a node with 2 Intel Xeon-E5 processors with 20 computing cores in total. The maximum and minimum speedup on 20 cores was 243 and 160.

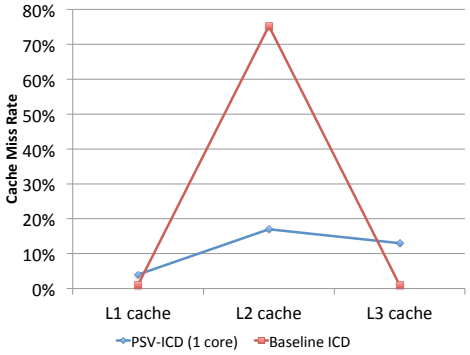
Importantly, by comparing column 3 with column 2, we can also see that almost 14.5 times speedup on average comes from the improved sequential code GFLOP efficiency E_F. This is directly a result of both the SV design and the SVB design, for the SV design increases the temporal locality of cache uses and the SVB design increases the hardware prefetching rate. Figure 3.7(a) shows L1, L2 and L3 cache level data cache miss rate in the same experiment. The L2 data cache miss rate decreases significantly from 75% to 17%, on average. Nevertheless, there is a minor cache miss increase at the L3 level because of the large amount of memory copies in steps 5, 6 and 14 in Algorithm 2. This trend is more obvious as the number of cores increases. Figure 3.7(b) shows the L1, L2 and L3 cache level data cache miss rates of PSV-ICD at different numbers of cores. With more cores and more memory copies, the L3 cache miss rate slowly increases from 13% at 1 core to 19% at 20 cores. However, at the same time, the existence of augmented SVB also decreases the L2 cache miss rate from 17% at 1 core to 10% at 20 cores.

In addition, the L2 cache prefetching also contributes to the L2 cache miss rate reduction. Baseline ICD has a 0.14% prefetch rate, and a 3.6% prefetch hit rate for

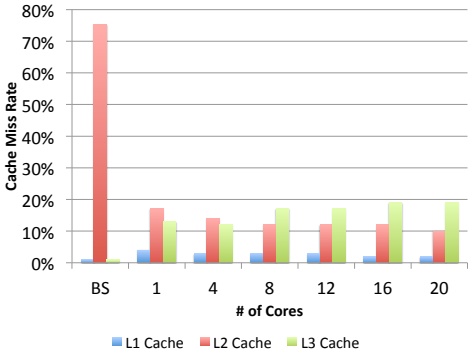
its L2 cache prefetcher on average. PSV-ICD (20 cores) L2 prefetch rate increases to 0.94% and its prefetch hit rate is 33.6% on average. We can observe that both the prefetch rate and the prefetch hit rate significantly improve when using SVBs. L1 and L3 prefetch rates are not measured because the Intel Haswell micro-architecture has no event type that can directly measure them.

These reductions in the data cache miss and the increase in the prefetch hit rate have a direct effect on E_F, the processor floating point operations efficiency. If we compare column 2 and column 3 in Table 3.1, E_F increases by 14.5 times on average but E_F drops somewhat when the number of cores increases. At 20 cores, E_F is 9 times higher than the baseline ICD's despite the lost parallel efficiency from multi-core communication cost and waiting on locks. For serial Baseline ICD, there is no multi-core communication cost or lock waiting overhead. At 20 cores, however, PSV-ICD spends 53% its time on lock waiting overhead. To help us understand the single core performance better, Figure 3.7(c) shows E_F at different numbers of cores when the multi-core communication cost and lock waiting overhead are not accounted for. We can see that PSV-ICD maintains good single core performance when the number of cores increases.

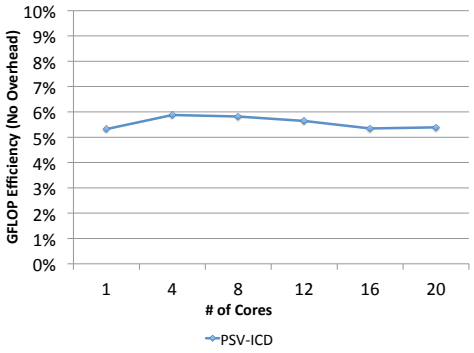
Results showing PSV-ICD parallel computations strong scaling speedup can be found in Figure 3.7(d). The important result of this figure is that the speedup is almost linear in the number of cores. This shows that PSV-ICD with augmented SVB is an efficient method for parallelizing the updates of multiple SVs. PSV-ICD has an advantage in both cache locality and lock wait overhead over the baseline ICD. In various experiments on PSV-ICD speedup, SV size has a significant fundamental impact on the speedup. Figure 3.7(e) records PSV-ICD speedup at different square SV side lengths. Since a SV is a group of voxels in the shape of a square in the image space, a SV side length of 9 corresponds to a SV with 81 voxels, for example. From Figure 3.7(e), we can see that SV side length of 13 has the best performance at 20 cores with a speedup of 187. Both increasing SV and decreasing the SV size will hamper its parallel performance. A too large SV size will definitely increase cache



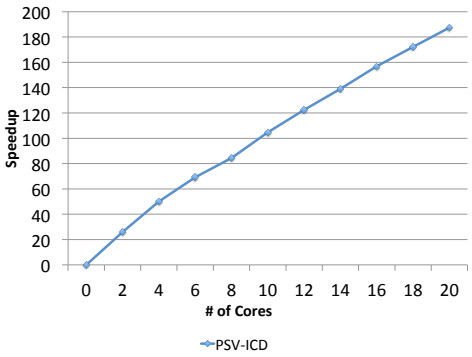
(a) PSV-ICD (1 core) and Baseline ICD's data cache miss rate at different levels of cache.



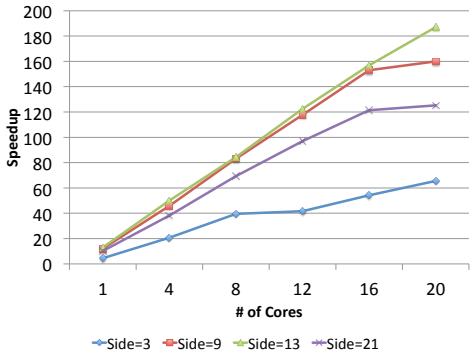
(b) PSV-ICD data cache miss rate up to 20 cores. BS is baseline ICD.



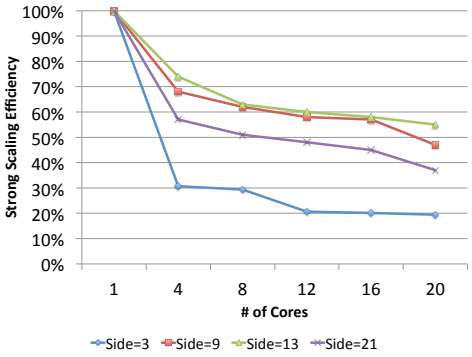
(c) PSV-ICD's average GFLOP efficiency given no communication overhead or lock wait time.



(d) PSV-ICD's parallel performance strong scaling speedup up to 20 cores.



(e) PSV-ICD strong scaling speedup at different square SV side lengths.



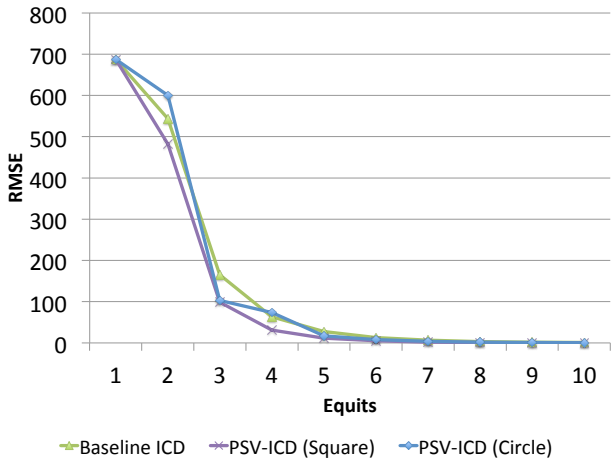
(f) PSV-ICD strong scaling efficiency at different square SV side lengths.

Fig. 3.7.

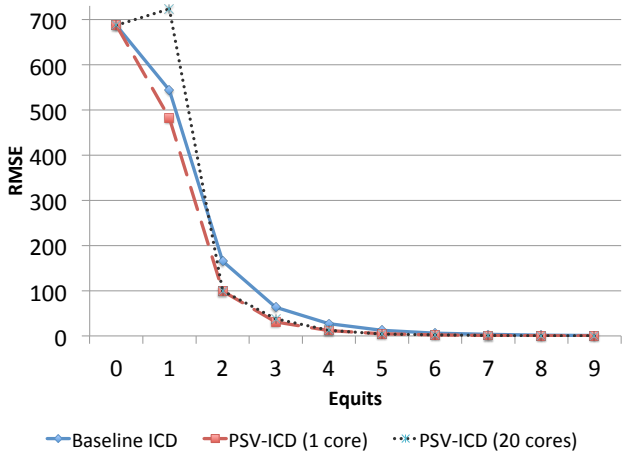
pressures and reduce the cache hit rate. A too small SV size, however, increases parallel communications frequency and lock waiting overhead in step 10 of Algorithm 4. VTune analyzer shows that PSV-ICD with a SV side length of 9 spends more than 72% of its time on waiting for locks when running on 20 cores. PSV-ICD with SV side length of 13, however, is able to reduce lock waiting time to 53% when running on 20 cores.

Figure 3.7(f) shows PSV-ICD strong scaling parallel efficiency performance at different SV side lengths. The numerator in the efficiency calculation is the sequential version of PSV-ICD. The denominator is the parallel version of PSV-ICD multiplied by the number of cores. PSV-ICD with SV side length of 3 drops its efficiency quickly because of waiting on locks when the number of cores increases. This also explains why GCD and PSV-ICD have such dramatic performance differences in parallel computations. GCD can be viewed as Inter-SV parallelism with only one voxel in each SV. Having a too small SV size will dramatically increase the lock wait time when the number of cores increases. On the other hand, PSV-ICD with a SV side length of 21 reduces the lock wait time but has a much steeper parallel efficiency drop compared with a SV side length of 13 because of the reduced cache hit rate from the large SV size. In addition, Figure 3.7(f) also accounts for the efficiency lost from parallel executions. It takes about 5 seconds from 1 core sequential code to 1 core parallel code because PSV-ICD has the parallel atomic operations at step 10 in Algorithm 4, which the sequential code does not have. That explains why the parallel efficiency drops faster in the first few cores and then drops slower for more cores.

The PSV-ICD algorithm's convergence rate is also a point of interest. In this benchmark of 3200 test cases, we require all algorithms to converge fully, so as to have a better understanding of their convergence properties. Our results indicate that our PSV-ICD can converge to a RMSE of 8 HU after 4.1 equits on average with the benchmark data set. In addition, we did not observe a significant change in the convergence rate for PSV-ICD versus the baseline ICD algorithm. Figure 3.8(a) shows a plot of RMSE versus equits for PSV-ICD (1 core). For the two phase convergence



(a)



(b)

Fig. 3.8.: (a) Illustrates that it is better to use square SVs than circular SVs in the two phase convergence mechanism because square SVs converge faster. (b) Convergence speed plot of RMSE versus equits. PSV-ICD (20 cores) converges faster than Baseline ICD since the second equit.

mechanism, we note that convergence is slower when using circular SVs than with square SVs. Square SVs generally maintains the same convergence speed as the baseline ICD algorithm although it converges faster in the initial equits.

Figure 3.8(b) shows the convergence speed of baseline ICD and PSV-ICD with different cores. Note that PSV-ICD in Figure 3.8(b) uses square SVs. The results show that PSV-ICD's convergence speed (1 core) is no slower than the baseline ICD implementation and it even has a small edge in the initial equits. This advantage continues until after the fifth equit. As to the convergence plot of PSV-ICD (20 cores) algorithms, it is easy to see that the PSV-ICD algorithm initially suffers from overshoot due to the violation of dependencies discussed in the challenges of parallelism in Section 3.4. The robust convergence of MBIR, however, allows the PSV-ICD algorithm to self correct this inconsistency error in later equits. Starting from the second equit, the PSV-ICD algorithm converges as fast as SV-ICD. In addition, the convergence speed is not adversely affected by more parallel-computing units shown in the plot. PSV-ICD has the same convergence speed at 1 core and 20 cores.

3.6 CHAPTER CONCLUSION

While MBIR provides high quality image reconstructions and is agnostic to scanner geometries, it has been considered impractical because of its long execution time. ICD is able to speed up convergence but parallelism has been viewed as being very limited and the data layout has made effective use of cache memories difficult. In this work, we have shown different levels of parallelism available in high performance CT reconstruction and that inter-SV parallelism can be effectively used by the PSV-ICD algorithm. Our experimental results have shown significant speedups and reduced running time by using PSV-ICD, making MBIR practical. We view this as a major breakthrough in making MBIR available for a wide range of applications.

We can also note that we are still utilizing less than 6% of processor efficiency despite significant performance gains. Thus, the next chapter, Chapter 4, will focus

not only on scaling these applications to a large number of processors by utilizing other levels of parallelism, but also towards further increasing processor efficiency.

4. MASSIVELY PARALLEL 3D IMAGE RECONSTRUCTION

4.1 INTRODUCTION

The previous chapter discusses how high performance MBIR can be applied to a single-slice 2D image reconstruction on a single computer node with multi-cores. In this chapter, we will extend the high performance MBIR techniques discussed in the previous chapter to fully 3D image reconstruction on a supercomputer with tens of thousands of cores.

As discussed before in Chapter 1 and Chapter 3, MBIR [4, 5] is a regularized reconstruction method, based on Bayesian estimation. Extensive research in the past has shown that MBIR provides higher image quality than other methods [4, 5, 9]. However, MBIR's improved image quality comes at the cost of greater computation by several orders of magnitude, as compared to traditional methods [18, 28].

As summarized before in Chapter 3, algorithms for MBIR image reconstruction can be summarized into two categories: global update methods and local update methods, with each having advantages and disadvantages. For global update methods [22, 23], all voxels in a volume are updated simultaneously in every iteration. Therefore, the global update methods are natural for parallelism and enable good scalability. However, massive parallelism also leads to slow convergence [4, 18], sometimes requiring hundreds of iterations [17, 27].

In contrast, local update methods, such as coordinate descent [4, 29, 30, 38–41], have much faster convergence with fewer iterations [28] and allow for the simultaneous update of a small groups of voxels [30, 42–44]. For 3D image reconstruction, there are two state-of-the-art implementations that are based on coordinate descent:

Time-Interlaced MBIR (TIMBIR)[1] [9, 11] and Parallel Super-Voxel Iterative Coordinate Descent (PSV-ICD) [18, 45] (presented and discussed in Chapter 3) with each having its own advantages and disadvantages. TIMBIR parallelizes the updates for a small group of spatially separated voxels that share little data in common. These spatially separated voxel updates reduce synchronizations among cores. However, it also reduces cache locality and limits the scalability. Alternatively, PSV-ICD groups spatially close voxels that share much data. Then, a core updates a group of spatially close voxels in sequence with cache locality benefit. In the meanwhile, different cores update spatially separated groups to reduce synchronization overhead (or atomic operation overhead). PSV-ICD, however, is a single node implementation without scalability. Therefore, implementations for coordinate descent trade-off between cache locality and parallelism.

In addition to the trade-offs mentioned above, all implementations for MBIR suffer from irregular data access patterns and inefficient SIMD operations. For each voxel update, data must be accessed from sinogram, following an irregular pattern in computer memory. Importantly, this access pattern is different for each voxel, so there is no single reorganization of the data that can resolve this SIMD inefficiency.

In summary, the four major issues that must be addressed to improve MBIR's computational performance are: (1) data access pattern regularity, (2) cache locality, (3) parallelism, and (4) convergence speed.

This chapter addresses the four challenges to 3D MBIR performance by presenting a massively parallel algorithm, *Non-Uniform Parallel Super-Voxel* (NU-PSV). The NU-PSV algorithm utilizes *Three-Dimensional Super-Voxels* (3DSVs) to ensure good cache locality, low parallel overhead, and to retain the fast convergence of a local update method. It also utilizes a *Block Transposed Buffer* (BTB) data layout to completely linearize the per-voxel memory accesses and enable SIMD operations. In summary, this chapter has the following contributions:

- Introduce the concept of the 3DSVs for 3D image reconstruction;

[1]TIMBIR can be downloaded at `https://github.com/adityamnk/timbir`.

- Describe the BTB design that linearizes data accesses and improves SIMD operations;
- Provide experimental results showing that the NU-PSV algorithm scales to 69632 cores with a speedup of 1665 compared to PSV-ICD and 9776 compared to TIMBIR.

In Section 4.2, we review background information about the state-of-the-art implementations. Then, we discuss how the *Non-Uniform Parallel Super-Voxel* algorithm (NU-PSV) overcomes the performance challenges of MBIR. Section 4.3.1 discusses how NU-PSV regularizes data accesses and improves SIMD vectorization and prefetching. Section 4.4 discusses the parallelization of NU-PSV, and Section 4.5 discusses the improved algorithmic convergence. In Section 4.6, we present three reconstruction performance evaluations for NU-PSV on datasets from Air Force Research Lab, Lawrence Berkeley National Lab and the Department of Homeland Security. The Significant advances in the design and implementation of a high-performing, scalable 3D MBIR results in the NU-PSV algorithm scaling to 69632 cores on the Cori Xeon-Phi Knights Landing cluster and 1665 times faster than the state-of-the-art implementations. In Section 5.2, we discuss how NU-PSV can be applied to compressive sensing problems in general. Finally, we present the conclusions in Section 4.7.

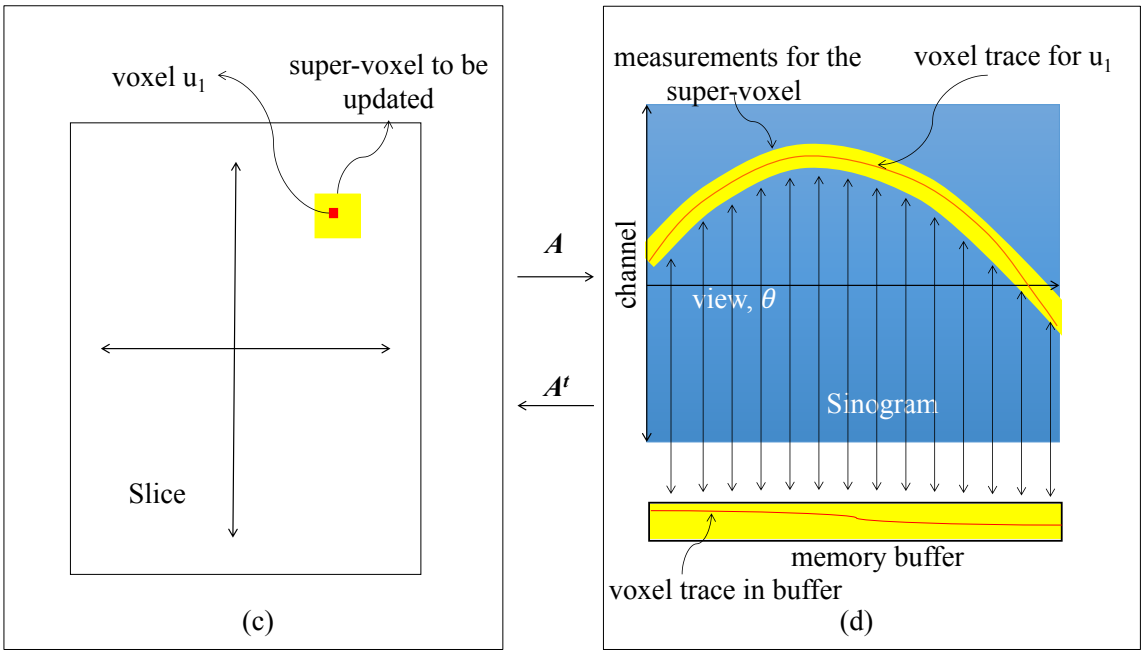


Fig. 4.1.: (a) Shows a super-voxel in a slice. (b) Measurements for the super-voxel follow a sinusoidal band in sinogram and measurements for u_1 fall within the sinusoidal band. After copying measurements from sinogram to a memory buffer, the voxel trace for u_1 still curves up and down, albeit with a smaller amplitude.

4.2 THE STATE-OF-THE-ART IMPLEMENTATIONS

In this section, we will discuss the state-of-the art implementations for 3D image reconstruction. And then, we will discuss their advantages and disadvantages.

The 3D MBIR reconstruction is computed as the solution to the following optimization problem:

$$\hat{x} = \underset{x \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(y - Ax)^T \Lambda (y - Ax) + R(x) \right\} , \tag{4.1}$$

where *argmin* is the minimizer that returns the image reconstruction, $\hat{x}$; y is a vector of length M, whose elements are the measurements in the sinogram; x is a vector of size N, whose elements are voxels of the volume. A is a sparse system matrix of size $M \times N$, representing the scanner system geometry, and each element $A_{i,j}$ roughly corresponds to the length of intersection between the j^{th} voxel and the i^{th} projection. If the i^{th}

projection measurement intersects the j^{th} voxel, then $A_{i,j}$ will be a non-zero value. Otherwise, $A_{i,j}$ will be zero. Λ is a pre-computed diagonal weight matrix, containing the inverse variance of the scanner noise; and $R(x)$ is a regularizing function.[2] The first term of the Equation (3.1), given by $\frac{1}{2}(y - Ax)^T\Lambda(y - Ax)$, is known as the data-fitting term because it penalizes the function when the reconstruction, $\hat{x}$, fits poorly with the measurements, y. The second term, $R(x)$, is a regularizing prior function that penalizes the function when x has irregular spatial properties [4].

To compute the solution to Equation (3.1), coordinate descent methods sequentially updates voxels or, in some cases, may update groups of voxels in parallel [29, 30, 42, 43]. Algorithm 5 summarizes the key operations that must be performed to update a voxel, x_j.

Algorithm 5 Voxel Update (j, x, e, Λ, A)

1: $\theta_{1,j} \leftarrow -e^T\Lambda A_{*,j}$

2: $\theta_{2,j} \leftarrow A_{*,j}\Lambda A_{*,j}$

3: $\alpha \leftarrow \text{argmin}_{\alpha \geq -x_j} \left\{\theta_{1,j}\alpha + \frac{1}{2}\theta_{2,j}\alpha^2 + R(x_j + \alpha)\right\}$

4: $x_j \leftarrow x_j + \alpha$

5: $e \leftarrow e - A_{*,j}\alpha$

The error term, e, equals to $y - Ax$; $A_{*,j}$ is the j^{th} column of A matrix; α is the change of x_j's due to the update; and $R(x_j + \alpha)$ is the prior function for x_j, which depends upon the voxels neighboring to x_j. Typically, the neighboring voxels to x_j are the 8 nearest voxels in the same slice and 2 nearest voxels in the adjacent slices. In Algorithm 5, steps 1 and 2 calculate the first and second derivatives required for step 3. In step 3, α is chosen to minimize the cost function. In step 4, voxel x_j is updated. In step 5, the error term is updated so that it is consistent with the updated voxel value, x_j. In typical applications, the computation of Algorithm 5 is dominated by steps 1 and 2 since each voxel typically intersects many projections. This set

[2] In fact, Equation (3.1) represents the general form for most of the compressive sensing problems [18, 45]. Therefore, its solution is of general interest (see Section 5.2).

of projections that must be indexed correspond to the sinusoidal path illustrated in Figure 2.4.

Below we review the two state-of-the-art implementations of coordinate descent reconstruction, TIMBIR and PSV-ICD. TIMBIR has poor cache locality, low parallel overhead and limited scalability. PSV-ICD, however, has good cache locality, relatively low parallel overhead and no scalability.

TIMBIR For TIMBIR, each core updates all the voxels in a single slice, and different cores update different slices of the volume in parallel. The benefit of TIMBIR is that slices of the volume are spatially separated. Therefore, TIMBIR has low synchronization overhead, but it also has poor cache locality. TIMBIR, however, provides limited scalability. Since the updates of adjacent slices have dependency (see the prior function $R(x_j + \alpha)$ in Algorithm (5)), only non-adjacent slices can be updated in parallel.

PSV-ICD To solve Equation (3.1), PSV-ICD reconstructs a volume slice by slice. For each slice, PSV-ICD groups spatially contiguous voxels into a *super-voxel*. Then, a core updates all voxels within a super-voxel in order. Since voxels within a super-voxel are spatially close, updates benefit from cache locality. Different cores, however, update spatially separated super-voxels in the same slice. Therefore, synchronizations (implemented as atomic operations) among cores are reduced. Figure 4.1(a) shows an example of a super-voxel. The measurements of the super-voxel follow the yellow sinusoidal band in the sinogram shown in Figure 4.1(b). Every voxel of the super-voxel, such as u_1, has a voxel trace inside the sinusoidal band. Therefore, voxels of a super-voxel often share measurements and cache locality can be better exploited.

To further improve cache locality, measurements for a super-voxel are copied from the sinogram to a memory buffer. Then, every voxel trace in the memory buffer, shown as a yellow rectangle in Figure 4.1(b), is flattened. Note that the voxel trace for u_1 becomes much more linear in the memory buffer, so that hardware prefetching can be improved. However, the voxel trace still curves up and down in the memory buffer, albeit with a small amplitude.

Despite the benefits outlined above, PSV-ICD has two major disadvantages: the data access pattern and scalability. Since the voxel trace still curves up and down in the memory buffer, efficient SIMD operations are not possible. In addition, PSV-ICD is a single node implementation which requires fine-grained atomic operations in a shared memory system. Therefore, PSV-ICD is not capable of large-scale parallelism.

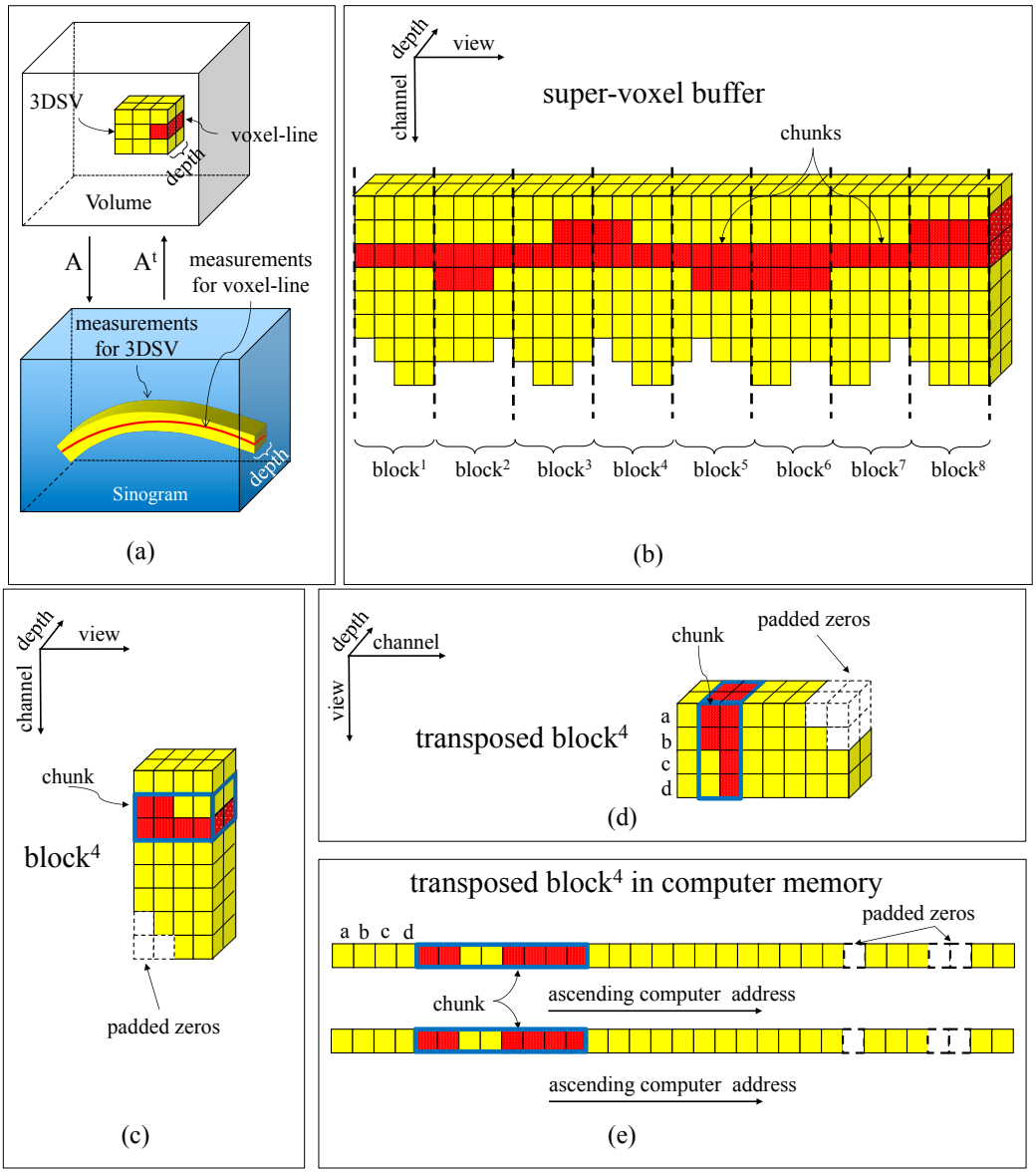


Fig. 4.2.: (a) Shows a 3DSV of size $3 \times 3 \times 2$. A voxel-line in the 3DSV is shaded in red. In addition, measurements for the 3DSV follow a sinusoidal band in sinogram. (b) Shows a super-voxel buffer. Notice that measurements for the red voxel-line trace curves up and down in the super-voxel buffer with a small amplitude. (c) shows $block^4$ of the super-voxel buffer with padded zeros. (d) Shows the transposed $block^4$. (e) Shows the memory layout for the transposed $block^4$. Notice that measurements in the chunk are grouped together in memory.

4.3 IMPROVING DATA ACCESS

In this section, we discuss how we improve SIMD vectorization and prefetching by regularizing data access. In addition, subsection 4.3.1 proposes the 3DSV and BTB designs and subsection 4.3.2 analyzes the potential performance gain by using the BTB design.

4.3.1 BLOCK TRANSPOSED BUFFER

We define a group of voxels in the shape of a rectangular cuboid as a 3DSV. Figure 4.2(a) shows a 3DSV of size $3 \times 3 \times 2$, where 3 is the 3DSV width and height, and 2 is the 3DSV depth. The sequence of voxels along the depth (z axis) is called a *voxel-line* [12, 28]. Figure 4.2(a) shows an example of voxel-lines, in which the 3DSV has 9 voxel-lines in total and each voxel-line contains 2 voxels. One such voxel-line is shown and shaded in red within the 3DSV.

Figure 4.2(a) illustrates the measurements associated with a 3DSV as a yellow sinusoidal band in the sinogram. The measurements for a single voxel-line within the 3DSV are then shown as a red trace within the sinusoidal band. Notice that both the sinusoidal band and the red voxel-line trace are three-dimensional with depth 2. Since each voxel-line must access measurements from the sinusoidal band, the measurements in the sinusoidal band will be repeatedly accessed as voxel-lines are updated. Therefore, the 3DSV design retains the cache locality benefit of the PSV-ICD algorithm.

In order to further improve cache locality and enable more effective hardware prefetching, the 3DSV's measurements can be copied from the sinusoidal band to a memory buffer, called the *super-voxel buffer*. Figure 4.2(b) shows the layout of a super-voxel buffer for the 3DSV in Figure 4.2(a). Since the sinusoidal band is three-dimensional, the corresponding super-voxel buffer also has three dimensions (channel, view, and depth), with measurements laid out along the channel direction in memory, i.e., adjacent channel entries are adjacent in memory.

Within the super-voxel buffer, all measurements for a voxel-line, also shaded in red in Figure 4.2(b), are accessed along a sinusoidal trace with depth 2, and with a much smaller amplitude than in the sinogram. Nevertheless, the combination of remaining trace amplitude, varying trace width (as explained in Section 2.2), and varying super-voxel buffer width (as in Figure 4.2(b)) still lead to irregular data access. In order to have efficient SIMD operations, the data access must follow a linear data access pattern, i.e., the memory access pattern follows the form $ik_1 \pm jk_2 \pm k_3$ for a linear sequence of is and js. To achieve such a goal, the super-voxel buffer must have constant trace width, constant buffer width and no amplitude.

Before we present the technical solution, we start by defining a series of reorganized data structures. We start by defining a *block* to be a fixed number of contiguous views within a super-voxel buffer, and denote the i^{th} block as $block^i$. For the example shown in Figure 4.2(b), each block consists of 4 views of the super-voxel buffer, and the entire super-voxel buffer has 8 blocks in total.

In addition, we define a *chunk* as a rectangular cuboid, circumscribing the sinusoidal voxel-line trace in a block. Figure 4.2(c) illustrates $block^4$ of the super-voxel buffer, and the chunk contained within $block^4$ is outlined with a bold blue line. When processing a voxel line, all the measurements in a chunk will be accessed even though only a portion of them are needed. We will see that it is more efficient to access the entire chunk because the width of the chunk is constant. Of course, the chunk will typically contain some measurements that will not be required for the voxel-line update. These measurements that are unrelated to the voxel-line update are shown in yellow in a chunk. We define these unneeded measurements in a chunk as *redundant measurements*. The measurements that are required for the voxel-line update, however, are shown in red in a chunk, and we define these required measurements in a chunk as *essential measurements*.

In addition, each block is padded with zeros so that the buffer width, namely the number of channels of a block, is constant. Figure 4.2(c) illustrates this zero padding in which six zeros are added to the bottom of $block^4$, making $block^4$ a rectangular

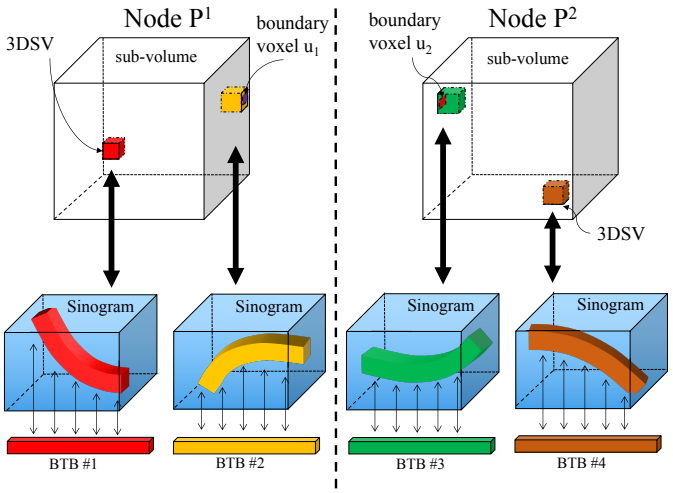
cuboid. Since the buffer and the voxel-line trace widths are constant, the data access for a voxel-line update is completely linear in each block. Although the addition of redundant measurements moderately increase cache pressure by adding unneeded data, the performance gain from the linear data access far outweighs the loss from increased cache pressure.

Although redundant measurements and zero padding linearize data access, measurements in a chunk have non-contiguous memory locations and data access requires gather-scatter overhead, prohibiting SIMD performance. To further improve SIMD performance and have more contiguous memory access, we define the *Block Transposed Buffer* (BTB) as a block-wise *(i.e., block-by-block)* transposition of each block in the super-voxel buffer. Figure 4.2(d) illustrates the transpose of $block^4$, with the axes of channel and view swapped by the transposition. Notice that the BTB is now a regular cuboid that is stored in the order of view, channel, and depth. More specifically, all measurements within the block are laid out along the view direction in memory. For example, measurements *a, b, c, d* in Figure 4.2(d) have contiguous memory locations. As shown in Figure 4.2(e), after transposition the memory accesses for the chunk in BTB^4 fall into two groups, outlined by bold blue lines, and each group have completely contiguous memory locations. Hence, the pattern of reads and writes for a voxel-line update now allow a high level of SIMD vectorization. Figure 4.6(c)-(e) in Section 4.6 gives a detailed evaluation of the block design, showing that the block design delivers a SIMD speedup of approximately 6.5.

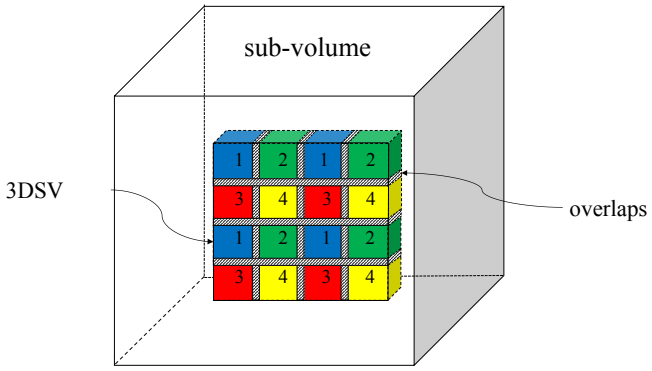
We now show analytically the effectiveness of the BTB in increasing SIMD instructions. In a BTB, the average number of completely linear memory accesses in a block, denoted N_{run}, is approximately given by

$$N_{run} = (N_{wh}C_1N_b^2 + C_2N_b)N_d \tag{4.2}$$

where N_{wh} is the number of voxels along the width and height of the 3DSV, N_b is the block size, N_d is the number of voxels along the depth of the 3DSV, C_1 and C_2 are constants related to the dataset and image size, whose detailed analytical expressions



(a)



(b)

Fig. 4.3.: (a) Shows a volume equally distributed between nodes, P^1 and P^2. Each 3DSV in the sub-volume accesses its measurements efficiently from the private BTBs. In addition, the update of two adjacent boundary voxels u_1 and u_2 depends on each other. (b) Shows that each tile in the checkerboard pattern has 4 colors (numbered as 1-4) and the overlaps among neighboring 3DSVs are shaded gray.

are included in Section 4.3.2. In this equation, we can observe that SIMD performance gains can be optimized by increasing N_b and N_d. Experiments in Figure 4.6(a)-(d) show how SIMD performance can be optimized with increased block size and 3DSV depth.

Although the BTB design increases SIMD instructions, the redundant measurements in the BTB lead to additional computations. Among measurements in chunks, the percentage of essential measurements, denoted as E_c, can be computed as

$$E_c = \frac{C_2}{N_{wh}C_1N_b + C_2} \tag{4.3}$$

where Section 4.3.2 provides the in-depth derivation of Equation (4.3). From this equation, we can observe that E_c is inversely proportional to N_b. Therefore, increasing block size leads to excessive computations, but also more linear memory accesses. To corroborate Equation (4.3), experiment in Figure 4.6(e) shows the inverse relationship between E_c and block size.

4.3.2 THEORETICAL ANALYSIS FOR BTB

This section shows how Equations (4.2) and (4.3) are derived. Figure 4.4 shows a chunk in a BTB, where the voxel-line trace is colored in red and redundant measurements are colored in yellow. Suppose that the chunk length is $N_b\Delta_\beta$, where N_b is the block size, β is the view angle, ranging from 0 to π, and Δ_β is the view angle spacing; the chunk width is $N_d\Delta_x$, where N_d is the depth of the 3DSV and Δ_x is the voxel width; the chunk height is $L_{pw} + m(\phi)N_b\Delta_\beta$, where L_{pw} is the average voxel-line trace width, α is the angle between the voxel-line trace and the view direction, $m(\phi)$ is the average voxel-line trace absolute slope. Relating to Figure 4.4, L_{pw} equals to the length of EF, $m(\phi)N_b\Delta_\beta$ equals to the length of FG, Then, $L_{pw} + m(\phi)N_b\Delta_\beta$ equals to the chunk height, EG. With the dimensions of the chunk, the volume of the chunk, denoted as V_c, can be computed as:

$$V_c = N_b\Delta_\beta N_d\Delta_x(L_{pw} + m(\phi)N_b\Delta_\beta) \tag{4.4}$$

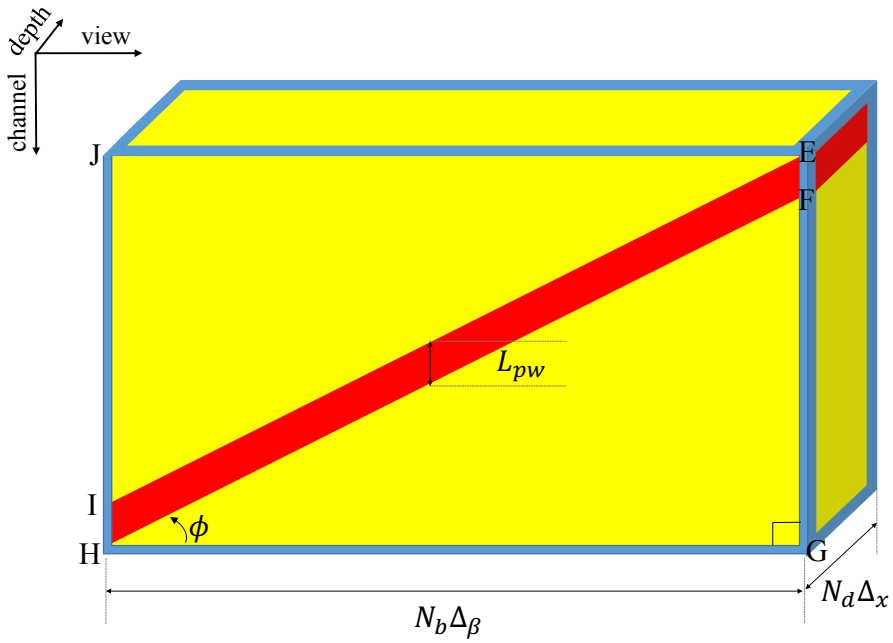


Fig. 4.4.: Demonstrates the dimensions of a chunk. The voxel-line trace is colored in red and the redundant measurements are colored in yellow.

Then, the average number of consecutive memory accesses for a block, denoted as N_{run}, can then be computed as:

$$N_{run} = \frac{V_c}{\Delta_\beta \Delta_x \Delta_d} \tag{4.5}$$

where Δ_d is the detector channel spacing, $\Delta_\beta \Delta_x \Delta_d$ is the size of a single entry in the BTB, $\frac{V_c}{\Delta_\beta \Delta_x \Delta_d}$ represents the number of element entries in a chunk. By plugging the expression of V_c to Equation (4.5), we can then get:

$$N_{run} = \frac{N_d N_b (L_{pw} + m(\phi) N_b \Delta_\beta)}{\Delta_d} \tag{4.6}$$

In Equation (4.6), N_d and N_b are known constants chosen to optimize computing performance, N_v, Δ_β and Δ_d are also known constants about the dataset. L_{pw} and $m(\phi)$, however, are parameters, whose computations are included in Appendices A and D.

After plugging Equations (A.3) and (D.5) into Equation (4.6), we can then get:

$$N_{run} = (\frac{N_{wh} \Delta_x N_b \Delta_\beta}{3\pi \Delta_d}(1 + \sqrt{2} + \ln(1 + \sqrt{2})) + \frac{4\Delta_x}{\pi \Delta_d} + 1) N_d N_b \tag{4.7}$$

Since $\Delta_\beta = \frac{\pi}{N_v}$, the full analytical expression for N_{run} becomes:

$$N_{run} = (\frac{N_{wh} \Delta_x N_b}{3\Delta_d N_v}(1 + \sqrt{2} + \ln(1 + \sqrt{2})) + \frac{4\Delta_x}{\pi \Delta_d} + 1) N_d N_b \tag{4.8}$$

If we let constant $C_1 = \frac{\Delta_x}{3\Delta_d N_v}(1 + \sqrt{2} + \ln(1 + \sqrt{2}))$ and constant $C_2 = (\frac{4\Delta_x}{\pi \Delta_d} + 1)$, then N_{run} can be simply stated as follows:

$$N_{run} = (N_{wh} C_1 N_b^2 + C_2 N_b) N_d \tag{4.9}$$

The percentage of essential measurements, E_c, can then be computed as the ratio of essential measurement entries to N_{run}. The number of essential measurements, i.e. the measurements for the voxel-line trace, can be computed as $N_{vl} = (\frac{4\Delta_x}{\pi \Delta_d} + 1) N_d N_b$, where N_{vl} is the number of essential measurements. E_c can then be computed as:

$$E_c = \frac{N_{vl}}{N_{run}} = \frac{C_2}{N_{wh} C_1 N_b + C_2} \tag{4.10}$$

4.4 IMPROVING PARALLELISM

NU-PSV exploits three orthogonal levels of parallelism that effectively utilize large parallel machines:

Intra-SV parallelism: data-level parallelism (SIMD vectorization) across multiple voxels within a 3DSV;

Inter-SV parallelism: parallelism across multiple 3DSVs in a sub-volume, where a sub-volume is defined as a set of contiguous slices;

Intra-volume parallelism: parallelism across sub-volumes in a 3D volume.

Intra-SV parallelism. Intra-SV parallelism updates groups of voxels in a 3DSV in parallel using SIMD vectorization. An ideal choice for such groups is voxel-lines because the BLB design, discussed in the previous section, enables a linearized access pattern and a high level of SIMD vectorization. To update voxels, the intra-SV parallelism must cycle through the BTB blocks and compute $\theta_{1,j}$ and $\theta_{2,j}$ for each voxel. Since $\theta_{1,j}$ and $\theta_{2,j}$ are formed from dot products of vectors (see Algorithm 5), the values of $\theta_{1,j}$ and $\theta_{2,j}$ can be computed as summations of the partial results, $\theta_{1,j}^{\sim}$ and $\theta_{2,j}^{\sim}$, for each transposed block. Once the final versions of $\theta_{1,j}$ and $\theta_{2,j}$ are computed, they can be used to update the j^{th} voxel as in step 3 of Algorithm 5.

Algorithm 6 summarizes how the intra-SV parallelism functions for a 3DSV's update. The algorithm uses three BTBs denoted by BTB_e, BTB_Λ, BTB_A) to store the sinogram error, e, the sinogram weights, Λ, and the corresponding system matrix entries, $A_{*,j}$. The intra-SV parallelism exploits SIMD operations by assigning the computations for each transposed block to SIMD units in step 4. Each SIMD unit computes the partial result for a voxel, and different SIMD units compute results for different voxels. The summation of the computations at each transposed block is then used for the voxels' update in step 11. In this algorithm, voxels are processed in parallel through SIMD units, and the SIMD unit's access is completely linearized by the organization of the BTBs as discussed in 4.3.1. Therefore, the intra-SV parallelism efficiently utilizes the processor's SIMD units.

Algorithm 6 3DSV Update (U, x, BTB_e,BTB_Λ, BTB_A)

INPUT: U: the 3DSV to be updated

INPUT: BTB_e, BTB_Λ, BTB_A: the block-transposed memory for U

1: **for** each voxel-line, $V \in U$ **do**
2: Initialize $\theta_{1,j}$ and $\theta_{2,j}$ for each voxel, $j \in V$.
3: **for** each transposed block, tb **do**
4: **for** each voxel, $j \in V$, **do in parallel among SIMD units**
5: Compute $\theta^{\sim}_{1,j}$ and $\theta^{\sim}_{2,j}$, as in steps 1 and 2 of Algorithm 5
6: $\theta_{1,j}$ += $\theta^{\sim}_{1,j}$
7: $\theta_{2,j}$ += $\theta^{\sim}_{2,j}$
8: **end for**
9: **end for**
10: **for** each voxel, $j \in V$ **do**
11: $\alpha \leftarrow \text{argmin}_{\alpha \geq -x_j} \left\{ \theta_{1,j}\alpha + \frac{1}{2}\theta_{2,j}\alpha^2 + R(x_j + \alpha) \right\}$, as in step 3 of Algorithm 5
12: $x_j \leftarrow x_j + \alpha$
13: $BTB_e \leftarrow BTB_e - A\alpha$
14: **end for**
15: **end for**

The intra-SV parallelism, however, is difficult to implement because the BLB design mixes measurements for different voxels in a transposed block. Therefore, the intra-SV parallelism needs a special mechanism to differentiate measurements for different voxels, essential measurements, and redundant measurements.

For the parallel-beam CT scanner geometry, measurements for different voxels on the same voxel-line do not overlap with each other, making data differentiation among voxels easy. In the example of Figure 4.2(a), the voxel-line has two voxels. The chunk of computer memory locations for the first voxel's measurements are all in the first row of Figure 4.2(e). The chunk of computer memory locations for the second voxel's measurements are all in the second row of Figure 4.2(e), independent from the first row. Therefore, measurements for different voxels can be differentiated easily by accessing different computer memory locations without overlap.

To differentiate essential measurements and redundant measurements, we exploit the sparse A matrix structure for this purpose. For each essential measurement, its corresponding A matrix element is non-zero. Otherwise, the A matrix element is zero. In the example of Figure 4.2(e), essential measurements (shown in red within chunks) corresponds to non-zero A matrix elements. The redundant measurements (shown in yellow within chunks) corresponds to zero-value A matrix elements. Therefore, the computations for $\theta_{1,j}$ and $\theta_{2,j}$, as in steps 1 and 2 of Algorithm 5, are not negatively influenced by accessing redundant measurements, since the product of zero-value A matrix elements and redundant measurements are zero. Hence, the computational correctness for $\theta_{1,j}$ and $\theta_{2,j}$ are guaranteed.

Inter-SV parallelism. Inter-SV parallelism updates, in parallel, different 3DSVs within a sub-volume using the cores in a node. Algorithm 7 summarizes how the inter-SV parallelism functions. In steps 3 and 4, each 3DSV's core creates private BTBs for its 3DSV, U. Figure 4.3(a) shows an example of private BTBs. In step 6, the core follows Algorithm 6, updating voxel-lines in U. In step 7, the private BTB_e is transposed back to the super-voxel buffer, Buf_e.

At the end of a 3DSV's update, the updated measurements in the each core's super-voxel buffer, Buf_e, must be combined and written to the full sinogram. The updated measurements, however, overlap with one other, because voxels share measurements in the sinogram (see Section 2.2). Therefore, simply copying the updated measurements back to the sinogram will be lost and overwritten [18,45]. To correctly write measurements to the sinogram, $Buf_{e,\Delta}$ keeps track of all changes of the measurements in step 8. Then, $Buf_{e,\Delta}$ is atomically added back to the sinogram in step 9, so that the changes of measurements for other simultaneously updated 3DSVs are not lost and overwritten.

Inter-SV parallelism, however, leads to image artifacts. Since a sub-volume is artificially partitioned into 3DSVs, image artifacts are present along the 3DSVs' boundary [18, 45]. To address this, neighboring 3DSVs overlap with each other on their common boundary using halos, eliminating the partition boundary among neighboring 3DSVs. Figure 4.3(b) shows an example of overlapping 3DSVs, with the halo regions shaded in gray.

Although overlapping neighboring 3DSVs removes image artifacts, this can lead to data races. This situation occurs when the same boundary voxel is simultaneously updated by cores in step 12 of Algorithm 6. To eliminate possible races, we employ a checkerboard pattern [45] for 3DSVs' updates. As illustrated in Figure 4.3(b), checkerboard pattern tessellates a volume into tiles, with each tile consisting of 4 3DSVs with different colors, with no two adjacent 3DSVs having the same color. Inter-SV parallelism cycles through the four colors in step 1 of Algorithm 7 and only 3DSVs with the same color are updated in parallel. Therefore, no voxel on a boundary is updated in parallel by more than one core.

Intra-volume parallelism. Intra-volume parallelism performs parallel updates on all sub-volumes across nodes, with the i^{th} node processing the i^{th} sub-volume (S^i). Figure 4.3(a) shows how sub-volumes are equally distributed to two nodes. Each node then processes its sub-volume by using Algorithm 7.

Algorithm 7 inter-SV Update (S)

INPUT: S: a sub-volume to be updated.

1: **for** Tile color t from 1 to 4 **do**
2: **for** 3DSV, $U \in S$ with color t, **do in parallel among cores**
3: create super-voxel buffers, Buf_e, Buf_Λ and Buf_A
4: Block-transpose super-voxel buffers to BTB_e, BTB_Λ and BTB_A
5: Make a temporary copy of Buf_e to $Buf_e^{'}$
6: 3DSV Update (U, S, BTB_e,BTB_Λ, BTB_A), as in Algorithm 6
7: Block transpose BTB_e to Buf_e
8: $Buf_{e,\Delta} \leftarrow Buf_e - Buf_e^{'}$
9: Atomic operation: $e \leftarrow e + Buf_{e,\Delta}$
10: **end for**
11: **end for**

Since the prior function for each voxel update in Algorithm (5) depends upon its neighboring voxels, the update of boundary voxels in S^i depends on the adjacent boundary voxels in S^{i-1} and S^{i+1}. An example of this dependency is shown in Figure 4.3(a), where boundary voxels u_1 and u_2 are adjacent to each other and their updates depend on each other. Because of the iterative nature of MBIR [18, 46], violating this dependence does not prevent convergence. Intuitively, intra-volume parallelism can be viewed as a special case of updating spatially close voxels. If the neighboring sub-volumes each only have 1 voxel, then intra-volume parallelism reduces to spatially close voxel update, which is known to converge [36, 47]. In situations where sub-volumes, such as S^1, do not have adjacent boundary voxels, we assume that the sub-volumes have imaginary adjacent boundary voxels with value zero.

To reduce the communication latency in exchanging boundary voxels of the sub-volumes, the intra-volume parallelism uses a two-data-buffer design. Each node, P^i, has two allocated data buffers, buffer^{-1} for message passing with P^{i-1} and buffer^1 for message passing with P^{i+1}. When neighboring nodes send boundary voxels to P^i, a copy of voxels is passed asynchronously to the data buffers. P^i can then access the needed voxels from its allocated data buffers.

4.5 IMPROVING CONVERGENCE

Algorithm 8 NU-PSV (S^i)

INPUT: S^i: the sub-volume to be processed by P^i.

LOCAL: A_g: a group of greedy update 3DSVs.

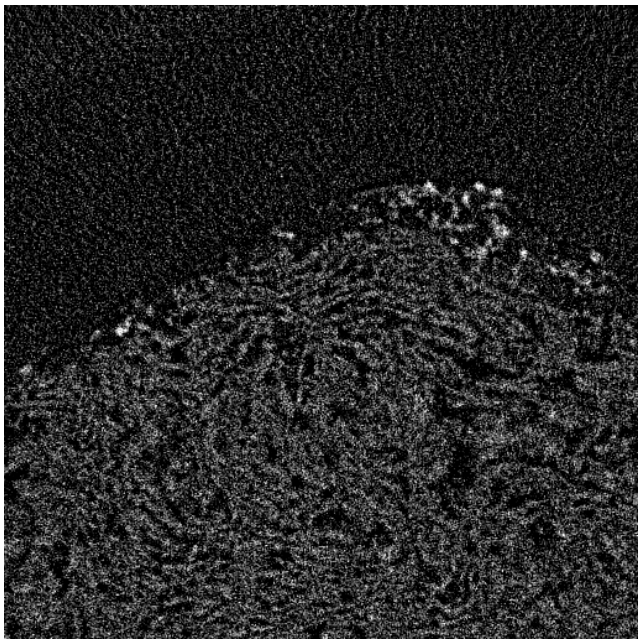
LOCAL: A_r: a group of random update 3DSVs.

inter-SV Update (S^i) as in Algorithm 7. A max-heap is constructed.
repeat
    Re-randomize 3DSVs and divide them into $\frac{C}{\rho}$ groups.
    for Group r from 1 to $\frac{C}{\rho}$ **do**
        Inter-SV Update (A_g), as in Algorithm 7.
        Inter-SV Update (A_r), as in Algorithm 7
        The max-heap is updated.
        Exchange boundary voxels asynchronously among P^i, P^{i-1} and P^{i+1}
    end for
until NU-PSV converges.

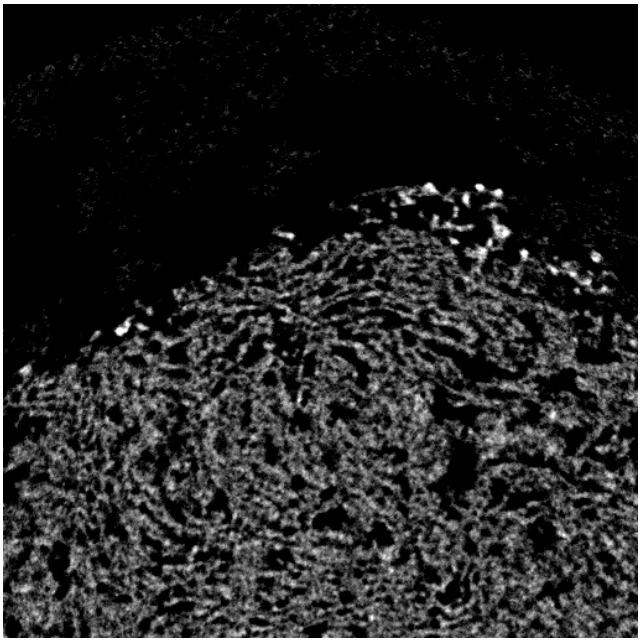
Although the checkerboard update pattern prevents races in inter-SV parallelism, it leads to slow convergence because the checkerboard update pattern enforces a cyclic and fixed update order, which typically has a poor convergence speed experimentally [46].

For fast convergence, previous theoretical work proves that a combination of random updates and greedy updates have the best algorithmic convergence speed [46]. We now discuss how we apply these theoretical findings to improve the convergence of the NU-PSV algorithm. Experiments in Section 4.6 show that the NU-PSV algorithm provides, on average, a 1.66 times speedup from faster convergence over the same algorithm with the random update procedure alone.

Intuitively, the NU-PSV algorithm needs random updates because the randomized order ensures that the 3DSVs to be updated are usually spatially separated, reducing



(a)



(b)

Fig. 4.5.: The upper image is a slice reconstructed by Filtered Back-Projection. The lower image is the same slice reconstructed by MBIR. Notice that MBIR has significantly better image quality with less noise.

the correlations among 3DSVs. However, the NU-PSV algorithm also needs greedy updates because 3DSVs in a volume require different frequencies of updates. For example, a high radio-density 3DSV, which represents material such as bones or metals, requires many more updates than a low-radio density 3DSV, which represents material such air.

Algorithm 8 describes the NU-PSV algorithm that uses the two update procedures. Step 1 updates all 3DSVs in the sub-volume in a randomized order. At the end of the updates, a max-heap is constructed, containing every 3DSV's absolute magnitude of change in the previous randomized update, with the top elements of the max-heap being the 3DSVs with the highest magnitude of change. In step 3, the NU-PSV algorithm randomly divides 3DSVs into $\frac{C}{\rho}$ groups, where each group contains $A_r = \frac{N\rho}{C}$ of the 3DSVs and N is the number of 3DSVs in the sub-volume being processed by node P^i. Parameter ρ is from 0 to 1 and controls the extent to which updates are in random update and are in greedy update. In particular, $\rho = 0$ causes NU-PSV to perform greedy updates only and $\rho = 1$ causes NU-PSV to perform random updates only.

In step 4, NU-PSV visits each randomized group in order. In each visit, the algorithm alternates between a greedy update procedure (step 5) and a random update procedure (step 6). In the greedy update procedure, NU-PSV processes the top $A_g = \frac{N(1-\rho)}{C}$ of the 3DSVs on the max-heap and updates them in parallel. After the greedy update procedure, all voxels in the visited randomized group is processed and updated in parallel. The random update procedure, however, makes the max-heap out of date. In step 7, the max-heap is updated with the new magnitude of change information from the random update procedure. Finally, each node P^i exchanges the boundary voxels with neighboring nodes P^{i-1} and P^{i+1}, as discussed in Sec. 4.4. In step 10, the NU-PSV algorithm repeats itself until algorithmic convergence is reached.

Table 4.1.: Performance attributes table summarizing how the experimental results are reported.

Attributes	Contents
Type of method	*Fully implicit*
Results reported on basis of	*Whole application without I/O*
Precision reported	*Single precision*
System scale	*Measured on full-scale system*
Measurement mechanism	*Timer*, *static analysis tool*

4.6 EXPERIMENTS

This section experimentally evaluates the NU-PSV algorithm, and shows how the techniques discussed in this chapter contribute to NU-PSV's performance.

Datasets. To measure NU-PSV's performance, I use three datasets: a sparse-view iron hydroxide (FeOOH) material dataset, imaged with an electron microscope at the United States Air Force Research Lab, a slime mold biology dataset (*m vesparium*), imaged using a synchrotron at the Lawrence Berkeley National Lab, and a security baggage scan dataset for a high cluttered volume, imaged with a security CT Scanner at the Department of Homeland Security. Each data point is computed three times and the average is used. Figure 4.5 shows an example slice in the iron hydroxide dataset, reconstructed by the traditional reconstruction method (Filtered Back-Projection) and MBIR respectively. With the same amount of sparse-view measurements, it can be seen that the MBIR reconstruction has less noise and better image quality than the traditional method.

Computing platform. Experiments were performed on Cori Xeon-Phi Knights Landing clusters [48] at The National Energy Research Scientific Computing Center (NERSC), using the Cray Aries network. Each node features a 96-GB memory, a 16-GB high bandwidth memory (cache mode), and a 68-core processor (processor model: knights landing 7250). Each core has a 32-KB private L1 data cache, and

every two cores share a 1-MB L2 cache. All programs were compiled with the Intel MPI compiler version 2017.up1 using the -O3 -qopenmp -xMIC-AVX512 compiler options. Intra-voxel parallelism is achieved using AVX-512 SIMD instructions. The Inter-SV parallelism is achieved using OpenMP, while intra-volume parallelism is achieved using MPI.

Tools. All runtimes are based on the entire program except for I/O that reads the input dataset and writes the outputs from/to NERSC global file system. The reported SIMD speedups in this section are the relative speedup between a program with SIMD vectorization and the exact same program without SIMD vectorization (using the -no-vec -no-simd compiler options). Cache miss rates are measured using the Intel Vtune Amplifier 2017.

Algorithmic parameters All experiments for NU-PSV use 3DSVs of size $15 \times 15 \times 8$, which was empirically determined to be the optimal size for the NERSC computer platform. The only exception is when a sub-volume has fewer than 8 slices and the depth of 3DSVs decreases proportionally. For example, if a sub-volume only has 1 slice, then the 3DSV size is $15 \times 15 \times 1$. The ρ and C parameters, discussed in Section 4.5, are empirically determined to be 0.2 and 3.25 respectively for faster convergence.

Performance attributes The performance attributes of the experimental results reported in this section are summarized in Table 4.1. This table shows that (1) all algorithms used in this section are fully implicit numerical methods; (2) all performance results are reported based on the whole application without I/O; (3) the computation precision reported in this section is single precision floating points; (4) the performance results are all measured on the full-scale system; and (5) the tools used for performance measurements are timer and static analysis tools.

Table 4.2.: Runtimes in seconds for the iron hydroxide dataset. The first row is the number of allocated nodes in the reconstruction and the second row is the number of allocated cores (each node has 68 cores). The third row of the table is the average runtimes for TIMBIR at different numbers of allocated cores. Note that TIMBIR can only scale to 544 cores. The fourth row is the average runtimes for PSV-ICD. The fifth row of the table is the average runtimes for NU-PSV.

Nodes	1 Node	4 Nodes	8 Nodes	64 Nodes	256 Nodes	1024 Nodes
Cores	68 Core	272 Cores	544 Cores	4352 Cores	17408 Cores	69632 Cores
TIMBIR	>86400.0	>86400.0	76252.9	NA	NA	NA
PSV-ICD	12988.4	NA	NA	NA	NA	NA
NU-PSV	1039.5	286.0	149.4	24.4	10.9	7.8

4.6.1 IRON HYDROXIDE DATASET

The reconstructed volume size for this dataset is 1024^3 with a voxel resolution of $64^3 \mu m^3$. The sinogram has: (1) parallel-beam sparse-view projections; with (2) uniform angle distribution from 0° to 180°; and (3) a sinogram size of 1024×225, where 1024 is the number of channels and 225 is the number of views.

To quantify the overall performance, NU-PSV's runtimes are compared to the performance of TIMBIR and PSV-ICD algorithms, discussed in Section 4.2. Table 4.2 summarizes the average runtimes of the three algorithms at different numbers of cores. In the third row, TIMBIR's runtimes at 1 and 4 nodes are more than the 24-hour wall time limit, noted as >86400 seconds in the table. Since TIMBIR can only scale up to 8 nodes for a 1024-slice volume (see Sec. 4.2), runtimes at more than 8 nodes are reported as NA. The fourth row is PSV-ICD's runtimes. Because PSV-ICD is a single node implementation, it only scales to 68 cores. The fifth row is NU-PSV, 9776.0 times faster than TIMBIR and 1665.2 times faster than PSV-ICD. NU-PSV's significant speedups come from two different sources: the per-node speedup (each node has 68 cores), with a speedup larger than 83.11 over TIMBIR and a speedup of 12.49 over PSV-ICDs, and the scalability, which scales to 69632 cores.

We now investigate the improved per-node performance. Figure 4.6(a) shows SIMD speedups at different 3DSV depths. The SIMD speedup is 1.21 with depth equal to 1 and the SIMD speedup peaks at 3.28 with depth equal to 8. This confirms Equation (4.2)'s analysis that a larger depth increases the number of linear memory accesses, leading to better SIMD performance. Then, the SIMD speedup decreases from 3.28 with depth 8 to 2.72 with depth 32 because most data can fit into cache at depth 8 and cache pressure is minimal. When the 3DSV depth is increased to 32, the increased depth proportionally increases the cache pressure, negating SIMD performance gains.

To validate the explanations discussed above, Figure 4.6(b) shows the average number of linear memory accesses, N_{run}, at different 3DSV depth. We can see that

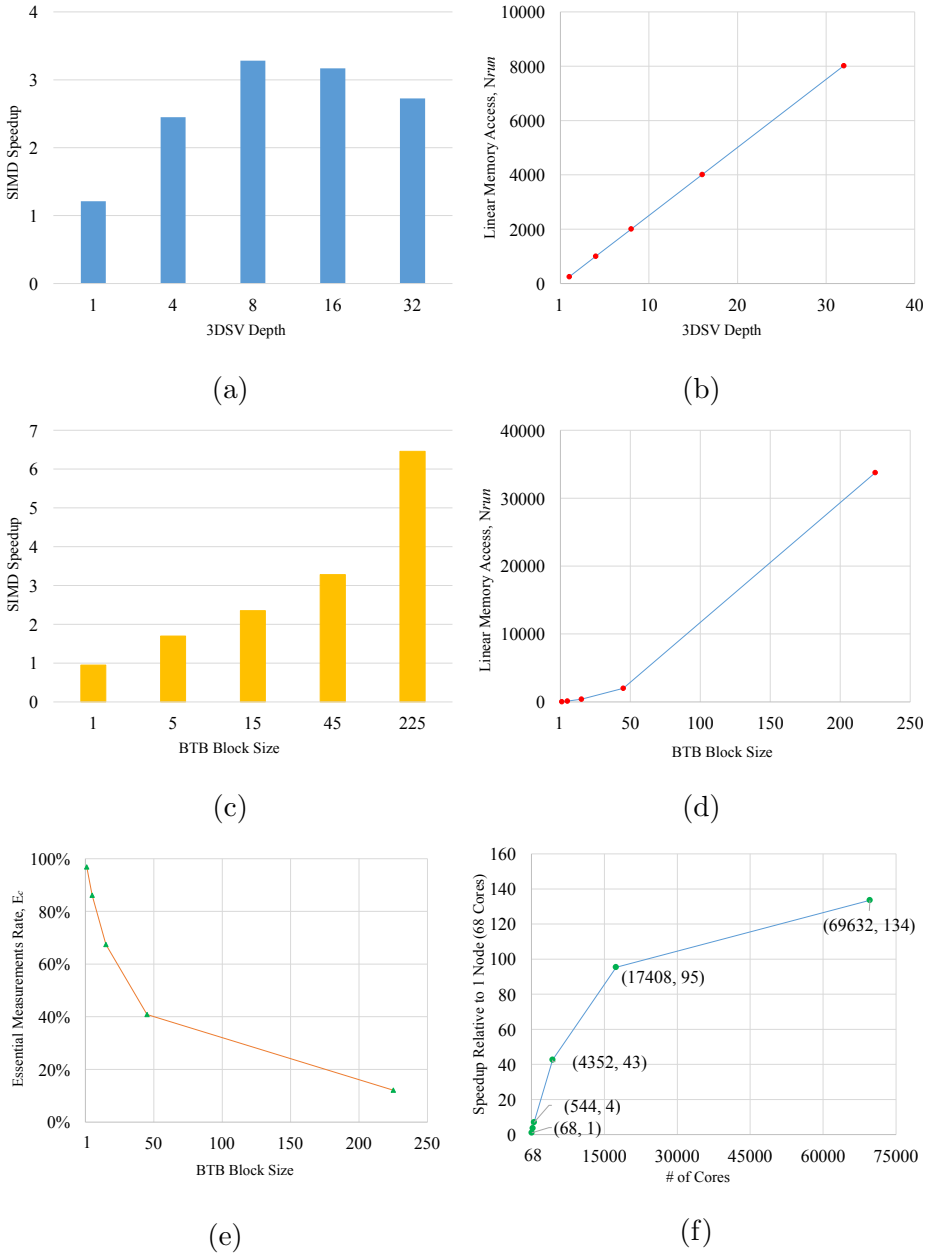


Fig. 4.6.: (a) The SIMD speedups at different 3DSV depths. (b) The average number of linear memory accesses, N_{run}, at different depths. (c) The SIMD speedups at different block sizes. (d) N_{run} at different block sizes. (e) The percentage of essential measurements, E_c, at different block sizes. (f) NU-PSV's speedup relative to 1 node (68 cores). The pairs at each data point indicate the speedup (second numbers of the pairs) at different numbers of cores (first numbers of the pairs).

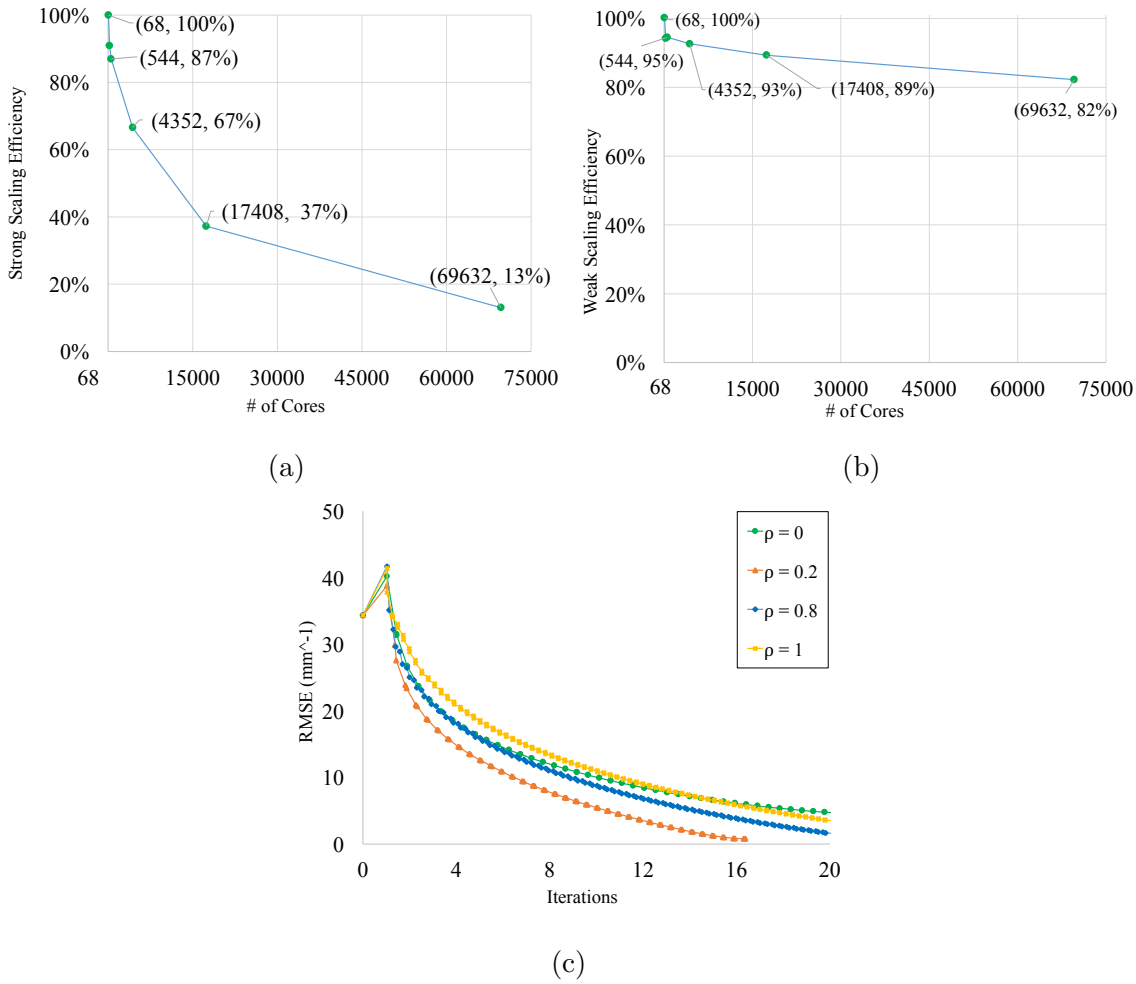


Fig. 4.7.: (a) Strong scalability of NU-PSV. The numbers at each data point indicate the strong scaling efficiency at different numbers of cores (efficiency baseline is 1 node). (b) Weak scalability of NU-PSV. (c) The algorithmic convergence speed for NU-PSV with different ρ.

N_{run} increases linearly with 3DSV depth. This observation verifies that 3DSV depth proportionally increases the number of linear memory accesses. In addition, the slope of the increase is large (slope equals to 250), indicating that a too large 3DSV depth can quickly worsen cache pressure and negatively impact the SIMD performance gains. To support the claim in the previous sentence, Intel Vtune Amplifier shows that the L2 cache miss rate increases from 6.4%, with depth equal to 8, to 32.3%, with depth equal to 32.

Another factor for NU-PSV's improved per-node performance is the BTB block size. Figure 4.6(c) shows the SIMD speedups at different block sizes. The SIMD speedup is 0.95 with block size equal to 1, and the SIMD speedup is 6.45 with block size equal to 225. This result confirms the theoretical analysis in Equation (4.2) that a larger block size increases N_{run}, leading to better SIMD vectorization.

Figure 4.6(d) corroborates claims in Figure 4.6(c), showing that the average number of linear memory accesses, N_{run}, increases proportionally with BTB block size. Therefore, an increased block size increases cache pressure. However, the increased block size does not increase the gather and scatter overhead while the increased depth does increase the overhead, explaining why the SIMD speedup is more scalable with block size than with 3DSV depth. Vtune Amplifier also supports this claim showing that the SIMD speedup is 3.28 and the L2 cache miss rate is 6.4% at block size 45. When the block size is increased to 225, the SIMD speedup is increased to 6.45, but with a lower L2 cache miss rate of 3.4%. The cache miss rate is lowered because the benefit from spatial locality outweighs the loss from compulsory cache miss when the block size becomes bigger.

Although a larger block size leads to more SIMD speedup, as indicated in Figure 4.6(c) and (d), a larger block size also increases redundant measurements, leading to extra computations. Figure 4.6(e) shows the relationship between the percentage of essential measurements, E_c, in a BTB and the block size. From the figure, We can observe that E_c is inversely proportional to block size. Therefore, there is a trade-off between SIMD performance gains and extra computations when choosing an optimal

block size. When the block size is smaller, each core has fewer linearized data in a block, leading to a smaller SIMD performance gain. However, a smaller block size also incurs fewer extra computations. In the example in Figure 4.6(c) and (e), block size 45 enables a SIMD speedup of 3.28 and E_c equals to 41%. When the block size is larger, each core has more SIMD performance gain, but also has more extra computations. In the example in Figure 4.6(c) and (e), block size 225 enables a SIMD speedup of 6.45 and E_c equals to 12%.

We now investigate NU-PSVs scalability. Figure 4.6(f) shows NU-PSV's scalability up to 69632 cores. The data points along the plot-line are NU-PSV's speedups relative to 1 node (68 cores) at different numbers of cores. For each data point, the first number in the pair represents the number of cores and the second number represents the speedup. We can observe that NU-PSV's speedup at 69632 cores (1024 nodes) is 133.6.

Figure 4.7(a) shows the strong scaling scalability of NU-PSV. The data points along the plot-line are the strong scaling efficiency at different numbers of cores. Note that the strong scaling parallel efficiency drops from 67% at 4352 cores to 13% at 69632 cores. The efficiency drop has two causes: (1) worse SIMD performance at a high number of cores and (2) a lower ratio of work to synchronization overhead. At 69632 cores (1024 nodes), each sub-volume has only one slice, restricting the 3DSV depth to 1. As explained in Figure 4.6(a), NU-PSV has worse SIMD performance at depth 1, resulting in lost parallel efficiency with a larger number of cores. In addition, a small 3DSV size resulting from high number of cores leads to less useful work per core. Therefore, NU-PSV has a lower ratio of work to synchronization overhead.

Figure 4.7(b) shows weak scalability of NU-PSV. The data points along the plot-line are the weak scaling efficiency at different numbers of cores. We observe that the weak scaling efficiency drops from 93% at 4352 cores to 82% at 69632 cores. Since the per-core work is fixed for weak-scaling experiments, parallel efficiency does not decrease because of worse SIMD performance or more frequent synchronizations. Therefore, the weak-scaling efficiency is much higher than the strong-scaling one.

Table 4.3.: Runtimes in seconds for the slime mold dataset.

Nodes	18 Nodes	80 Nodes	1080 Nodes	2160 Nodes
Cores	1224 Cores	5440 Cores	73440 Cores	146880 Cores
TIMBIR	>86400.0	NA	NA	NA
NU-PSV	1815.6	378.2	32.3	24.8

We now investigate how the alternating random and greedy updates affect convergence. Figure 4.7(c) shows the convergence at different values of ρ, where $\rho = 0$ (the green curve) is the result using greedy updates alone; $\rho = 0.2$ (the orange curve) is the result for 20% random updates and 80% greedy updates; $\rho = 0.8$ (the blue curve) is the result using 80% random updates and 20% greedy updates; and $\rho = 1$ (the yellow curve) is the result using random updates alone. Among different ρ values, $\rho = 0.2$ gives the best convergence rate, reaching full convergence in 16 iterations. From Figure 4.7(c), it is clear that the convergence is significantly faster when NU-PSV uses both of the random update and the greedy update procedures, and this conclusion agrees with previous results [28, 46].

4.6.2 SLIME MOLD DATASET

The slime mold dataset is 70 times larger than the iron hydroxide dataset and the reconstructed volume size is $2560 \times 2560 \times 2160$ with a voxel resolution of $1.316^3 \mu m^3$. The sinogram has: (1) parallel-beam projections; (2) uniform angle distribution from 0° to 180°; and (3) a sinogram size of 2560×1024, where 2560 is the number of channels and 1024 is the number of views.

Table 4.3 summarizes the runtimes for TIMBIR and NU-PSV. The single-node PSV-ICD is not included because one node has insufficient memory to hold the dataset. In Table 4.3, TIMBIR is only scalable to 1224 cores because of the al-

Table 4.4.: Runtimes in seconds for the security dataset.

Nodes	1 Nodes	4 Nodes	40 Nodes	440 Nodes
Cores	68 Cores	272 Cores	2720 Cores	29920 Cores
TIMBIR	45033.6	32035.2	NA	NA
PSV-ICD	1608.9	NA	NA	NA
NU-PSV	239.1	59.1	18.0	15.9

gorithmic constraint, discussed in Sec. 4.2, whereas NU-PSV scales to 146880 cores, with a total speedup >3483.9, compared to TIMBIR.

4.6.3 SECURITY DATASET

The security dataset is 3 times smaller than the iron hydroxide dataset and the reconstructed volume size is $512 \times 512 \times 440$, with a voxel resolution of $927.6^3 \mu m^3$. The sinogram has: (1) parallel-beam projections; with (2) uniform angle distribution from 0° to 180°; and (3) a sinogram size of 1024×720, where 1024 is the number of channels and 720 is the number of views.

Table 4.4 summarizes the reconstruction runtimes for TIMBIR, PSV-ICD and NU-PSV. TIMBIR is scalable to 272 cores because of the algorithmic constraint, discussed in Sec. 4.2. PSV-ICD is scalable to 68 cores. As a comparison, NU-PSV scale to 29920 cores with a speedup of 2014.8, compared to TIMBIR, and a speedup of 101.2, compared to PSV-ICD.

4.7 CHAPTER CONCLUSION

MBIR is a 3D reconstruction technique that has a large and growing impact on the scientific, medical and security imaging communities. Despite of its superior image quality, reconstruction time has been a barrier to its wider acceptance [18, 49, 50].

The technical innovations of this chapter will yield immediate results to the scientific imaging community. The slime mold dataset from the Lawrence Berkeley National Laboratory takes more than 200,000 NERSC hours for a single reconstruction and imposes a significant wall-clock delay in seeing the results of experiments. Our techniques, however, can reduce that with a 2 orders of magnitude improvement to 1860.5 NERSC hours at 73440 cores.

For the sparse-view iron hydroxide experiments, similar benefits will accrue. In addition, MBIR requires significantly fewer measurements than typical reconstruction methods. Therefore, NU-PSV also has the potential to improve the utilization of the imaging instrument.

In the security domain, bag scans require high image quality and a per-bag reconstruction time in less that 15 seconds. The NU-PSV algorithm is the only MBIR algorithm that is close to meeting these constraints.

The benefits of NU-PSV also apply to a wide range of compressive sensing problems, whose kernel algorithmic computations is Algorithm 5. These problems include the absolute shrinkage and selection operator (LASSO) problems in machine learning [51], geophysics sensing problems [52] and radar sensing problems [53]. All of these problems have irregular but predictable data layout, for which our techniques can provide improved cache, SIMD and prefetch utilization in addition to large-scale parallelism. More details on how the NU-PSV algorithm can be applied to compressive sensing problems in general can be found in Section 5.2.

5. THE BROADER APPLICATION TO OTHER DOMAINS

While Chapters 3 and 4 have presented and discussed techniques that lead to high-performance tomography, much is unknown how these techniques can be useful for the imaging community and the computing community in general. In this chapter, we will discuss the applications of these techniques and broaden their usage to the general sensing problems.

5.1 THE IMPLICATIONS OF THE PSV-ICD ALGORITHM

In Chapter 3, we focuses on the problem of high-performance 2D CT imaging systems. The PSV-ICD algorithm we describe in Chapter 3, however, is applicable to a broad range of sensing problems that can be expressed in the form

$$\hat{x} = \arg\min_{x} \{\|y - Ax\|_{\Lambda} + S(x)\} \tag{5.1}$$

where $\hat{x}$ is the sensor output data, y is measurement data, $S(x)$ is a stabilizing regularizer, Λ is a weighting matrix and A is an unstructured sparse matrix.

This sensor output data can be viewed as an image. In fact, most imaging problems can be put into the framework of equation (5.1). These include problems such as whole body CT, PET, or MRI imaging, transmission and scanning electron microscopy, synchrotron, neutron imaging, proton imaging and ultrasound imaging.

Nonetheless in many other cases, such as autonomous navigation, depth sensors, digital holography, graphical inference, geophysical sensing, radar, synthetic aperture radar, lidar, synthetic aperture lidar, radio astronomy, crystallography, machine learning techniques such as the least absolute shrinkage and selection operator, the data is simply a multidimensional array of quantitative measurements of the envi-

ronment or an object under test. For many of these problems, also known as the "compressive sensing problems," the objective is to sense some underlying state of the physical world from some indirect, noisy, sparse measurements and the stabilizing function, $S(x)$ in equation 5.1, is typically taken to be an L_0 or L_1 norm. In recent years, there has been a great deal of interest in compressive sensing as a method for extracting high fidelity data from sparse measurements [54].

For the general problem of equation (5.1), the PSV-ICD algorithm provides a multi-core parallel framework for computing the compressive sensing problems efficiently. The traditional approach has been to find columns of A that are uncorrelated [32]. More specifically, if we define the correlation between columns to be

$$cor(i,j) = \sum_{k=1}^{N} |A_{k,i}| \cdot |A_{k,j}| \tag{5.2}$$

then the traditional approach is to find different columns i and j such that $cor(i,j) = 0$. Intuitively, when $cor(i,j) = 0$, the columns i and j share no values of y in common and the processing of these columns may be performed independently. This approach has been driven by the desire to find columns which lead to "embarrassingly parallel" processing tasks [29, 30]. Columns that are uncorrelated, however, result in little or no memory reuse since both columns access completely different entries of the measurement data, y. This means that the resulting algorithm is severely bounded by the memory access.

In Chapter 3, we demonstrate that the PSV-ICD algorithm can achieve both parallelization and cache locality in the compressive sensing problems by breaking the common conviction. The approach of PSV-ICD is to select columns of i and j that maximize the value of $cor(i,j)$ for each core. At the same time, however, the value of $cor(i,j)$ for columns processed on different cores is minimized. In the framework of equation (5.1), an SV is a set of columns $\mathcal{S}$ such that for all $i,j \in \mathcal{S}$, $cor(i,j)$ is large. A large value of $cor(i,j)$ allows for a great deal of memory reuse since entries in y can be accessed many times for each core. This memory reuse can dramatically reduce cache miss rates and lead to much faster performance on a single

core. Across different SVs, the cross SV correlation, $cor(i, j)$, is small. A small value of $cor(i, j)$ across different SVs allows for more parallelism and fewer lock contentions. This balance between cache locality and parallelism is achieved through hierarchical parallelism, discussed in detail in Section 3.4.

5.2 THE IMPLICATIONS OF THE NU-PSV ALGORITHM

In addition to its application to 2D reconstruction, MBIR is also a widely-explored 3D image reconstruction technique that has a large impact on medical imaging, industrial imaging and scientific imaging. However, the slow computation speed for MBIR is a bottleneck for the scientific advancements in imaging community. Chapter 4 describes the NU-PSV algorithm that can significantly improve the computation speed for MBIR by regularizing data access pattern, reducing cache misses, enabling more parallelism and speeding up algorithmic convergence. Therefore, the computation performance for MBIR can be significantly improved. In addition to the contributions to the imaging community, the NU-PSV algorithm can also serve as a massively parallel framework for compressive sensing problems. In this section, we discuss how the NU-PSV algorithm can serve as a massively parallel framework.

As stated before in Section 5.1, most of the compressive sensing problems can be expressed as:

$$\hat{x} = \underset{x \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(y - Ax)^T \Lambda (y - Ax) + R(x) \right\} \tag{5.3}$$

Where x is the sensor output of size N, y is measurements of size M, A is a $M \times N$ system matrix that contains the physical properties of sensors, x is sensor output, Λ is a weighing matrix, and $R(x)$ is a stabilizing regularizer.

To apply the NU-PSV algorithm to compressive sensing problems, sub-volumes are processed in parallel among nodes and we define a sub-volume to be a cluster of A matrix columns, so that the statistical correlation among sub-volumes is minimal. In other words, if S_1 and S_2 are two clusters, their statistical correlation, $\sum_{A_{*,i} \in S_1, A_{*,j} \in S_2} \sum_{k=1}^{M} |A_{k,i}| \cdot |A_{k,j}|$, is minimized. Therefore, sub-volumes share min-

imal measurements in common and the inter-node synchronization overhead can be minimal.

In addition, 3DSVs are processed in parallel among cores and we define a 3DSV to be a subset of a sub-volume, so that the A matrix columns in the 3DSV has maximal statistical correlation, defined as $\sum_{k=1}^{M} |A_{k,i}| \cdot |A_{k,j}|$, where $A_{*,i}$ and $A_{*,j}$ are A matrix columns in the 3DSV. Since A matrix columns in the 3DSV has maximal statistical correlation, each core can maximize the cache locality benefit by reusing measurements.

To update 3DSVs in parallel, the two update procedures (random update and greedy update procedures discussed in Section 4.5) is used. In the random update procedure, a random group of 3DSVs is chosen and 3DSVs in the group are processed in parallel among cores. Since 3DSVs are chosen randomly, the statistical correlation among 3DSVs are moderately low, leading to a low synchronization overhead among cores. In the greedy update procedure, a group of "most-needed" 3DSVs, denoted as g, are updated in parallel, so that the gradient of this group, $|\nabla_g f(x)|$, is maximized. Therefore, the algorithmic convergence can also be maximized.

6. SUMMARY AND FUTURE WORK

In this dissertation, I discuss the application of high performance computing to CT image reconstruction. In recent decades, iterative reconstruction has become the standard of quality for the field because it has been demonstrated to greatly improve the image quality as compared to more traditional reconstruction algorithms. However, iterative reconstruction has a major challenge: it is known to be very computationally demanding. Therefore, iterative reconstruction is widely considered to be impractical and unsuited for commercialization because of its slow speed.

The research challenge for iterative reconstruction can be summarized as follows: (1) lack of cache locality; (2) difficulty in parallelization; (3) slow algorithmic convergence; and (4) irregular data accesses. Iterative reconstruction tends to have bad cache locality because the storage of imaging measurement data typically depends on the geometry of imaging equipment. Therefore, the data access pattern in iterative reconstructions is usually irregular, prohibiting good cache locality. In addition, iterative reconstruction is an example of stencil problems, so that each pixel update has a strong dependency with its neighboring pixels, but it also has a weak dependency with spatially distant pixels. Therefore, updating spatially neighboring pixels in parallel can exploit cache locality, but it increases parallel overhead and slows down the algorithmic convergence. Updating spatially separated pixels in parallel can ease parallel overhead and algorithmic convergence, but worsen cache locality. In addition, MBIR suffers from poor data regularity and algorithms for MBIR exhibit inefficient SIMD operations. Therefore, a high-performance solution for iterative reconstruction requires insights to imaging equipment, and parallel computing as well as numerical analysis.

To address the challenges discussed above, this dissertation develops two algorithms, the PSV-ICD algorithm and the NU-PSV algorithm to address them and

develop efficient innovations for 2D image reconstruction and 3D image reconstruction respectively.

2D CT Iterative Reconstructions on multi-core CPUs To reconstruct each voxel, the tomographic image reconstruction requires access to data following a sinusoidal pattern in a matrix, strongly prohibiting spatial locality and SIMD operations. Therefore in Chapter 3, this dissertation proposes the concept of the super-voxel (SV) to improve the locality and performance. A SV is a group of voxels in the image space in the shape of a square and whose measurement data is close together in the matrix. Therefore, cache miss rate can dramatically decrease because data are repeatedly used among voxels in a super-voxel, even though each voxels update still follows a sinusoidal path. Thus, operating on the data for these voxels is likely to increase temporal and spatial locality. In addition, Chapter 3 proposes the inter-SV parallelism to maintains both good cache locality and good parallel performance. To speed up convergence, inter-SV parallelism operate on multiple SVs simultaneously that are far apart in the image space. Each core of a node gets a SV to update and different cores update different SVs. These SVs are chosen to be spatially separated so that the distance between these SVs minimizes interference and lock contention resulting from simultaneous SVs update. Each SV has its own SVB, and within a SV updates are performed sequentially. Experiments show that the SV design and the inter-SV parallelism enables an average 187 times speedup comparing to the baseline ICD algorithm.

3D CT Iterative Reconstructions on massively parallel CPUs Chapter 4 extends the techniques developed in Chapter 3 to fully 3D image reconstruction. To map onto a hierarchical parallel computer architecture, each CPU updates a cross-section of the reconstructed volume. Each core of a CPU updates different SVs in the cross-section. In addition, Chapter 4 develops the concept of three-dimensional super-voxel (3DSV), which is a rectangular cuboid in the volume. By using 3DSV, the computations for MBIR can not only take advantage of the cache locality benefit, but also have completely regularized data access, enabling a high level of SIMD operations.

Beyond the concept of 3DSV, Chapter 4 also develops the Non-Uniform Parallel 3DSV Algorithm (NU-PSV) that applies a non-uniform update frequency to voxels in the volume so that algorithmic convergence can be improved. Extensive results indicate that the super-voxel algorithm has an average speedup of 9776 compared to the fastest state-of-the-art 3D image reconstruction implementation on a 69632-core distributed system.

Although NU-PSV algorithm has a large impact on MBIR and address its performance challenges, one of the remaining barriers is the automated selection for parameters. In NU-PSV algorithm, the performance related parameters (such as the selection of the 3DSV size and the block size in Section 4.3, the selection of the convergence parameters, ρ and c, in Section 4.5, and etc.) are chosen through extensive experiments for the best performance. The selection of these parameters not only depends on the dataset, but also depends on the computer architecture. For example, ρ and c change the convergence speed. The block size selection not only changes the SIMD performance but also the cache locality. The 3DSV size changes cache locality, parallel overhead, and also convergence speed! Therefore, the selections of these parameters cannot be easily automated because of the involved complexity. In practice, however, time-sensitive applications do not allow conducting extensive experiments to choose the best-performing parameters. Especially since the dataset for CT scans become larger and larger and the time for a full reconstruction can be in days, an exhaustive search for the best performing parameters is not feasible. Therefore, an essential future work is to propose a mathematical model for choosing the parameters, so that this model can accurately predict the best performing parameters in the NU-PSV algorithm. Then, the NU-PSV algorithm can be useful in real-time applications.

Another important future work is to apply the NU-PSV algorithm to other compressive sensing problems. In both Sections 5.1 and 5.2, I propose basic mathematical models in applying NU-PSV to other compressive sensing problems. To successfully implementing NU-PSV to these problems, however, require a careful investigation for

each case. For example, the algorithms for MRI image reconstruction is similar to CT. However, the data access pattern is drastically different because MRI has a very different imaging modality. Therefore, NU-PSV must be redesigned to adapt to the MRI data access pattern.

REFERENCES

REFERENCES

[1] impactscan.org, "A brief history of ct," http://www.impactscan.org/CThistory.htm, 2013.

[2] S. Hughes, "Ct scanning in archeology, computed tomography - special applications," in *InTech*, Oct 2011.

[3] A. R. Kherlopian, T. Song, Q. Duan, M. A. Neimark, M. J. Po, J. K. Gohagan, and A. F. Laine, "A review of imaging techniques for systems biology," *BMC Systems Biology*, vol. 2, no. 1, p. 74, 2008. [Online]. Available: http://dx.doi.org/10.1186/1752-0509-2-74

[4] K. Sauer and C. Bouman, "A local update strategy for iterative reconstruction from projections," *IEEE Transactions on Signal Processing*, vol. 41(2), 1993.

[5] J. B. Thibault, K. D. Sauer, C. A. Bouman, and J. Hsieh, "A three-dimensional statistical approach to improved image quality for multi-slice helical ct," *Medical Physics*, vol. 34(11), 2007.

[6] V. Hart, "The application of tomographic reconstruction techniques to ill-conditioned inverse problems in atmospheric science and biomedical imaging," in *All Graduate Theses and Dissertations*, 2012, p. 1354.

[7] K. A. Dines and R. J. Lytle, "Computerized geophysical tomography," *Proceedings of the IEEE*, vol. 67, no. 7, pp. 1065–1073, July 1979.

[8] J. Howard, "Vector tomography applications in plasma diagnostics," *Plasma Physics and Controlled Fusion*, vol. 38, no. 4, p. 489, 1996. [Online]. Available: http://stacks.iop.org/0741-3335/38/i=4/a=004

[9] K. A. Mohan, S. V. Venkatakrishnan, J. W. Gibbs, E. B. Gulsoy, X. Xiao, M. D. Graef, P. W. Voorhees, and C. A. Bouman, "TIMBER: A method for time-space reconstruction from interlaced views," *IEEE Transactions on Computational Imaging*, vol. 1, no. 2, pp. 96–111, June 2015.

[10] P. Jin, C. A. Bouman, and K. D. Sauer, "A method for simultaneous image reconstruction and beam hardening correction," in *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, 2013, pp. 1–5.

[11] J. W. Gibbs, K. A. Mohan, E. B. Gulsoy, A. J. Shahani, X. Xiao, C. Bouman, M. D. Graef, and P. Voorhees, "The three-dimensional morphology of growing dendrites," *Scientific Reports*, 2015.

[12] X. Wang, K. A. Mohan, S. J. Kisner, C. A. Bouman, and S. P. Midkiff, "Fast voxel line update for time-space image reconstruction," in *The 41st IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016.

[13] M. F. S. G. M. D'Ariano, M. G. A. Paris, "Quantum tomography," in *Advances in Imaging and Electron Physics*, vol. 128, 2003, pp. 205–308.

[14] DHS/ALERT, "Research and development of reconstruction advances in ct-based object detection systems," https://myfiles.neu.edu/groups/ALERT/strategic_studies/TO3_FinalReport.pdf, 2009.

[15] P. Jin, E. Haneda, C. A. Bouman, and K. D. Sauer, "A model-based 3d multi-slice helical ct reconstruction algorithm for transportation security application," in *Second International Conference on Image Formation in X-Ray Computed Tomography*, 2012.

[16] S. J. Kisner, E. Haneda, C. A. Bouman, S. Skatter, M. Kourinny, and S. Bedford, "Model-based ct reconstruction from sparse views," in *Second International Conference on Image Formation in X-Ray Computed Tomography*, June 2012, pp. 444–447.

[17] S. Degirmenci, D. G. Politte, C. Bosch, N. Tricha, and J. A. O'Sullivan, "Acceleration of iterative image reconstruction for x-ray imaging for security applications," in *Proceedings of SPIE-IS&T Electronic Imaging*, vol. 9401, 2015.

[18] X. Wang, A. Sabne, S. J. Kisner, A. Raghunathan, C. A. Bouman, and S. P. Midkiff, "High performance model based image reconstruction," in *21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2016.

[19] K. B. Mathieu, H. Ai, P. S. Fox, M. C. B. Godoy, R. F. Munden, P. M. de Groot, and T. Pan, "Radiation dose reduction for ct lung cancer screening using asir and mbir: A phantom study," *Journal of Applied Clinical Medical Physics*, vol. 15, no. 2, pp. 271–280, 2014. [Online]. Available: http://dx.doi.org/10.1120/jacmp.v15i2.4515

[20] E. A. Smith, J. R. Dillman, M. M. Goodsitt, E. G. Christodoulou, N. Keshavarzi, and P. J. Strouse, "Model-based iterative reconstruction: Effect on patient radiation dose and image quality in pediatric body ct," *Radiology*, vol. 270, no. 2, pp. 526–534, 2014. [Online]. Available: https://doi.org/10.1148/radiol.13130362

[21] F. D. Carlos, D. Gursoy, D. Ching, K. J. Batenburg, W. Ludwig, L. Mancini, F. Marone, R. Mokso, D. Pelt, J. Sijbers, and M. Rivers, "Tomobank: A tomographic data bank for computational science," 2017, available from http://tomobank.readthedocs.io/en/latest/source/data.html.

[22] N. H. Clinthorne, T. S. Pan, P. C. Chiao, W. L. Rogers, and J. A. Stamos, "Preconditioning methods for improved convergence rates in iterative reconstructions," *IEEE Transactions on Medical Imaging*, vol. 12(1), 1993.

[23] J. Fessler and S. D. Booth, "Conjugate-gradient preconditioning methods for shift-variant pet image reconstruction," *IEEE Transactions on Image Processing*, vol. 8(5), 1999.

[24] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Pittsburgh, PA: Carnegie Mellon University, 1994.

[25] C. Kamphuis and F. J. Beekman, "Accelerated iterative transmission ct reconstruction using an ordered subsets convex algorithm," *IEEE Transactions on Medical Imaging*, vol. 17(6), 1998.

[26] S. Basu and Y. Bresler, "$O(N^2 \log_2 N)$ filtered backprojection reconstruction algorithm for tomography," *IEEE Transactions on Image Processing*, vol. 9(10), 2000.

[27] B. D. Man, S. Basu, J.-B. Thibault, J. Hsieh, J. A. Fessler, C. A. Bouman, and K. Sauer, "A study of different minimization approaches for iterative reconstruction in x-ray ct," in *IEEE Nuclear Science Symposium*, vol. 5, 2005, pp. 2708–2710.

[28] Z. Yu, J.-B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh, "Fast model-based x-ray ct reconstruction using spatially nonhomogeneous icd optimization," *IEEE Transactions on Image Processing*, vol. 20(1), 2011.

[29] J. Zheng, S. S. Saquib, K. Sauer, and C. A. Bouman, "Parallelizable bayesian tomography algorithms with rapid, guaranteed convergence," *IEEE Transactions on Image Processing*, vol. 9(10), 2000.

[30] J. A. Fessler, E. P. Ficaro, N. H. Clinthorne, and K. Lange, "Grouped-coordinate ascent algorithms for penalized-likelihood transmission image reconstruction," *IEEE Transactions on Medical Imaging*, vol. 16(2), 1997.

[31] J. Fessler, *Analytical Tomographic Image Reconstruction Methods.* Ann Arbor, MI: University of Michigan-Ann Arbor, 2009.

[32] S. Ha and K. Mueller, "An algorithm to compute independent sets of voxels for parallelization of icd-based statistical iterative reconstruction," in *The 13th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 2015.

[33] J. E. Bowsher, M. F. Smith, J. Peter, and R. J. Jaszczak, "A comparison of osem and icd for iterative reconstruction of SPECT brain images," *Journal of Nuclear Medicine*, vol. 79(5), 1998.

[34] Z. Yu, J.-B. Thibault, C. Bouman, K. Sauer, and J. Hsieh, "Edge-localized iterative reconstruction for computed tomography," in *10th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 2009.

[35] C. Hoilund, "The radon transformation," http://mlsp.cs.cmu.edu/courses/fall2012/lectures/Carsten_Hoilund_Radon.pdf, 2007.

[36] J. A. Fessler and D. Kim, "Axial block coordinate descent (ABCD) algorithm for x-ray ct image reconstruction," in *11th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 2011.

[37] P. Jin, S. J. Kisner, T. Frese, and C. A. Bouman, "Model-based iterative reconstruction (MBIR) software for x-ray ct," November 2013, available from https://engineering.purdue.edu/ bouman/software/tomography/mbirct/.

[38] C. A. Bouman and K. Sauer, "A unified approach to statistical tomography using coordinate descent optimization," *IEEE Transactions on Image Processing*, vol. 5, no. 3, pp. 480–492, March 1996.

[39] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l_1-regularized loss minimization," in *28th International Conference on Machine Learning*, 2011.

[40] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012. [Online]. Available: http://dx.doi.org/10.1137/100802001

[41] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, Jun. 2015. [Online]. Available: http://dx.doi.org/10.1007/s10107-015-0892-3

[42] J. Fessler, E. Ficaro, N. Clinthorne, and K. Lange, "Fast parallelizable algorithm for transmission image reconstruction," in *IEEE Medical Imaging Conference*, San Francisco, CA, October 21-28 1995.

[43] K. Sauer, S. Borman, and C. Bouman, "Parallel computation of sequential pixel updates in statistical tomographic reconstruction," vol. 3, October 23-25 1995.

[44] J. Zheng, S. S. Saquib, K. Sauer, and C. A. Bouman, "Parallelizable bayesian tomography algorithms with rapid, guaranteed convergence," *IEEE Transactions on Image Processing*, vol. 9, no. 10, pp. 1745–1759, Oct. 2000.

[45] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff, "Model-based iterative ct image reconstruction on gpus," in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: ACM, 2017, pp. 207–220. [Online]. Available: http://doi.acm.org/10.1145/3018743.3018765

[46] J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke, "Coordinate descent converges faster with the gauss-southwell rule than random selection," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 1632–1641. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045292

[47] T. M. Benson, B. D. Man, L. Fu, and J. B. Thibault, "Block-based iterative coordinate descent," in *IEEE Nuclear Science Symposium Medical Imaging Conference*, Oct 2010, pp. 2856–2859.

[48] T. N. E. R. S. C. Center, "Cori intel xeon phi nodes," http://www.nersc.gov/users/computational-systems/cori/cori-intel-xeon-phi-nodes/, 2017.

[49] T. Bicer, D. Gürsoy, V. D. Andrade, R. Kettimuthu, W. Scullin, F. D. Carlo, and I. T. Foster, "Trace: A high-throughput tomographic reconstruction engine for large-scale datasets," *Advanced Structural and Chemical Imaging*, vol. Vol. 3(6), pp. 1–10, 2017. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5313579/

[50] L. Lerner, "Adventures of the first neuroscientist at argonne," https://www.anl.gov/articles/adventures-first-neuroscientist-argonne, 2016.

[51] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 408–415.

[52] J. F. Claerbout and F. Muir, "Robust modeling with erratic data," *Geophysics*, vol. 38, no. 5, pp. 826–844, 1973.

[53] L. Wu, P. Babu, and D. P. Palomar, "Cognitive radar-based sequence design via SINR maximization," *IEEE Transactions on Signal Processing*, vol. 65(3), 2017.

[54] Y. C. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2012.

APPENDICES

A. THE AVERAGE VOXEL TRACE WIDTH

We model a voxel to be a square with voxel width Δ_x, as shown in Figure A.1. Suppose that an X-ray rotating gantry rotates around the voxel and the $r-z$ plane is the X-ray detector plane. On one end of the rotating gantry is an X-ray source. On the other end is the X-ray detector and X-ray projection lines pass through the voxel make a projection on the detector. β is the CT system angle of rotation ranging from 0 to π, and $\delta_1(\beta)$, shown as a green bar in Figure A.1, is the length of projection on the X-ray detector at angle β. Then $\delta_1(\beta)$ can be computed as:

$$\delta_1(\beta) = \begin{cases} \sqrt{2}\Delta_x \cos(\frac{\pi}{4} - \beta), & \text{if } \beta \in [0, \frac{\pi}{2}) \\ \sqrt{2}\Delta_x \sin(\beta - \frac{\pi}{4}), & \text{if } \beta \in [\frac{\pi}{2}, \pi) \end{cases} \tag{A.1}$$

Unfortunately, Equation (A.1) is not ideal because it assumes that Equation (A.1) is a continuous function. In practice, however, $\delta_1(\beta)$ must be a multiple of of Δ_d. In addition, if a part of a channel receives projections, then the entire channel has to be counted in Equation (A.1)'s calculation. Therefore, Equation (A.1) must be adjusted to offset the error.

Figure A.2 shows why $\delta_1(\beta)$ must be adjusted for error. Suppose that the detector channel spacing, Δ_d, equals to the voxel length and height, Δ_x, the calculation in Equation (A.1) concludes that $\Delta_1(\beta) \in [0\sqrt{2}]$. However, the "actual" $\delta_1(\beta)$ has to be an integer (either 1, 2 or 3), much different from the solution in Equation (A.1). Depending on how channels are aligned, $\delta_1(\beta)$ can equal to 1 with $\beta = 90°$, shown in Figure A.2(a), or 2 with $\beta = 45°$, shown in Figure A.2(b), or 3 with $\beta = 45°$, shown in Figure A.2(c).

To offset the error, a constant term Δ_d is added to $\delta_1(\beta)$ to offset situations when parts of a channel receiving projections. Therefore, the adjusted $\delta_1(\beta)$ is computed as such:

$$\delta_1(\beta) \approx \begin{cases} \sqrt{2}\Delta_x \cos(\frac{\pi}{4} - \beta) + \Delta_d, & \text{if } \beta \in [0, \frac{\pi}{2}) \\ \sqrt{2}\Delta_x \sin(\beta - \frac{\pi}{4}) + \Delta_d, & \text{if } \beta \in [\frac{\pi}{2}, \pi) \end{cases} \tag{A.2}$$

Therefore, the average voxel trace width, denoted as L_{pw}, can then be computed as:

$$\begin{aligned} L_{pw} &= \frac{\int_0^{\pi} \delta_1(\beta)\, d\beta}{\pi} \\ &= \frac{\int_0^{\frac{\pi}{2}} (\sqrt{2}\Delta_x \cos(\frac{\pi}{4} - \beta) + \Delta_d)\, d\beta + \int_{\frac{\pi}{2}}^{\pi} (\sqrt{2}\Delta_x \sin(\beta - \frac{\pi}{4}) + \Delta_d)\, d\beta}{\pi} \\ &\approx \frac{4\Delta_x}{\pi} + \Delta_d \end{aligned} \tag{A.3}$$

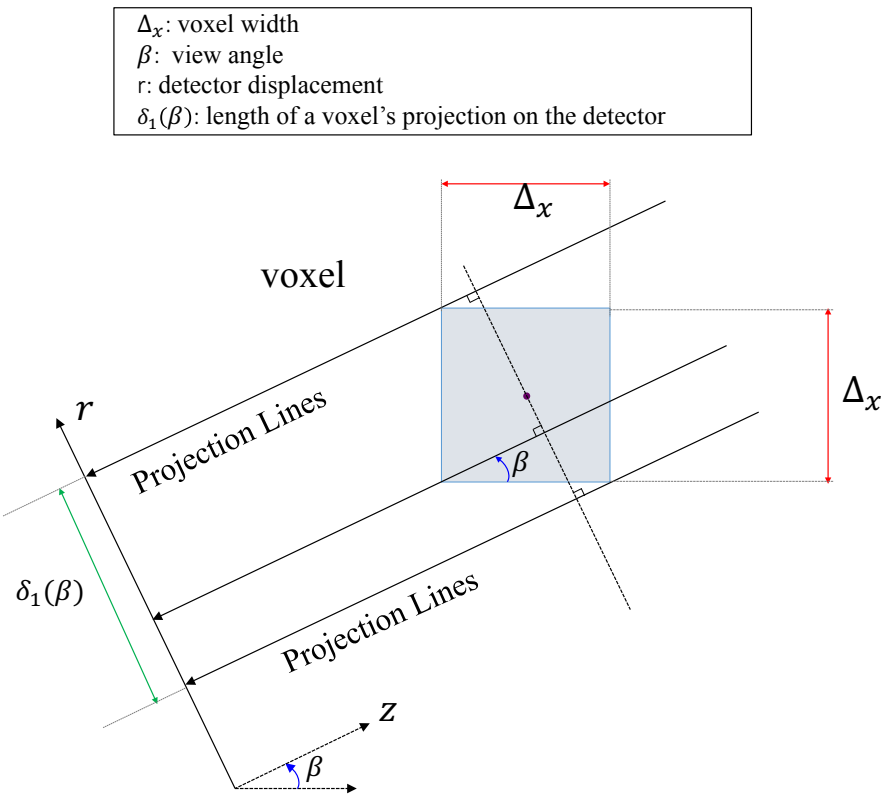


Fig. A.1.: Shows how a voxel is modeled as a square. $\delta_1(\beta)$ is shown as the green bar in the image.

For example, if $\Delta_x = \Delta_d = 1$, $\delta_1(\beta) \in [0, \sqrt{2}]$. In practice, however, $\delta_1(\beta)$ can only be integers 1, 2 or 3. See figure (a), (b) and (c).

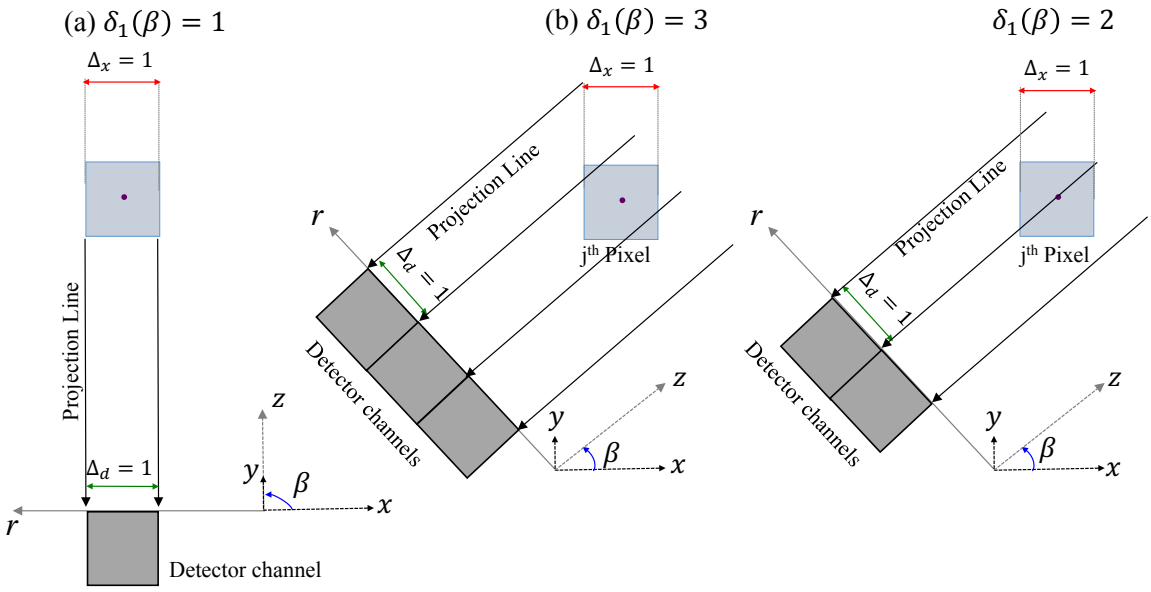


Fig. A.2.: Shows how $\Delta_1(\beta)$ is adjusted for error when $\Delta_x = \Delta_d = 1$. In (a), only one channel receives projections from the voxel when $\beta = 90°$. In (b), three channels receive projections from the voxel when $\beta = 45°$. In (c), two channels receive projections from the voxel when $\beta = 45°$.

B. THE AVERAGE SUPER-VOXEL BUFFER WIDTH

As defined in Section 3.3, a SV is a group of voxels in the shape of a square in the image space. Suppose that the number of voxels on each side of a square SV is denoted as N_{wh}, then the SV length and height are $N_{wh}\Delta_x$, where Δ_x is the voxel width for a single voxel. Figure B.1 shows an example of a SV, shaded in blue in the image space.

The measurements for the SV are grouped together in a sinusoidal band in the sinogram space, shaded in yellow in Figure B.1. To exploit hardware prefetching (see Section 3.3.2), measurements in the sinusoidal band can be copied to SVB, shown as a yellow rectangle in Figure B.1. To calculate the average SVB width, we can derive an equation by viewing a SV as a single voxel whose length and height equal $N_{wh}\Delta_x$. Replacing Δ_x in Equation (A.3) with $N_{wh}\Delta_x$, we can then derive an expression for the average SVB width, denoted as L_{SVB}, as such:

$$L_{SVB} \approx \frac{4N_{wh}\Delta_x}{\pi} + \Delta_d \tag{B.1}$$

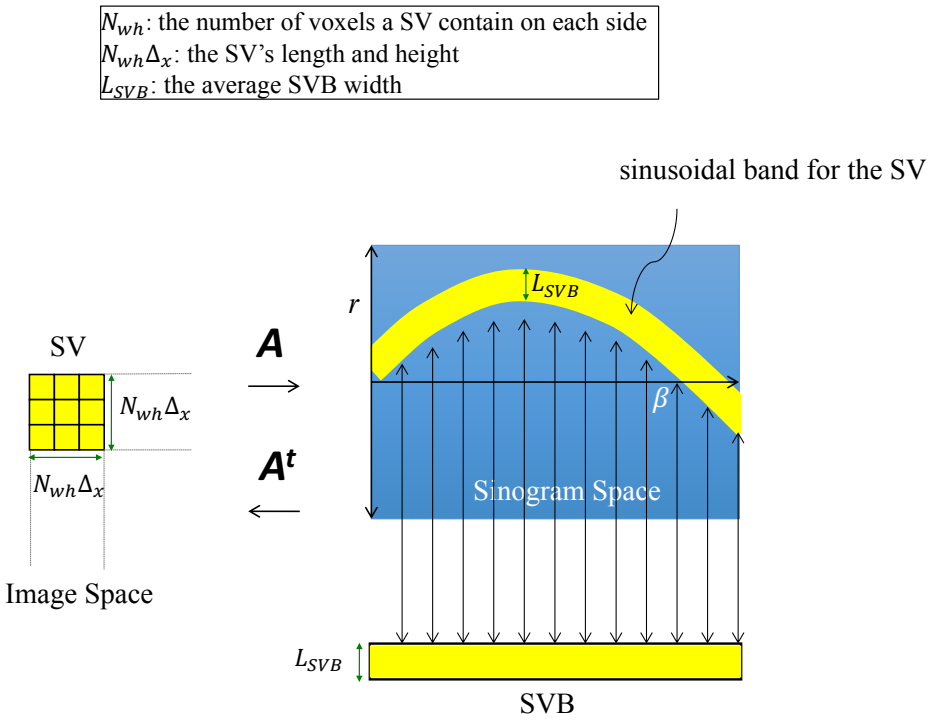


Fig. B.1.: Shows a SV in the image space whose length and height are $N_{wh}\Delta_x$. The measurements for the SV is a sinusoidal band whose average width is L_{SVB}. In addition, the sinusoidal band can be copied into a SVB with the same average width.

C. THE AVERAGE VOXEL TRACE AMPLITUDE

For the j^{th} voxel in the image space, we denote its position in the image space as (x_j, y_j). Figure C.1(a) depicts the j^{th} voxel in red. In addition, we know that the voxel's measurements correspond to a sinusoidal trace in the sinogram space (see Section 2.1). We use r_j to represent the amplitude of the sinusoidal trace in the sinogram space. In Figure C.1(b), the sinusoidal trace is shown in red in the sinogram space and r_j, shown by a yellow bar, represents the amplitude of the trace.

Analytically, r_j at view angle β can be expressed as:

$$r_j(\beta) = y_j \cos\beta - x_j \sin\beta \tag{C.1}$$

To facilitate the computation for the average amplitude of the sinusoidal trace, we denote the center of the image space with coordinate $(0, 0)$. In addition, we also assume that the image space has a square shape with length and height equal to N_x. Therefore, the four corners of the image space have coordinates $(\frac{N_x}{2}, \frac{N_x}{2})$, $(\frac{-N_x}{2}, \frac{N_x}{2})$, $(\frac{N_x}{2}, \frac{-N_x}{2})$, and $(\frac{-N_x}{2}, \frac{-N_x}{2})$.

Then, the average amplitude for a voxel trace in the sinogram space, denoted as $\tilde{A}$, can be computed as:

$$\tilde{A} = \frac{\int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_0^{\pi} |y_j \cos\beta - x_j \sin\beta| \, d\beta \, dx_j \, dy_j}{N_x N_x \pi} \tag{C.2}$$

To simplify the computations for Equation (C.2), we use polar coordinate to solve Equation (C.2) and we let $x_j = -\gamma \sin\beta$ and $y_j = \gamma \cos\beta$, where $\gamma = \sqrt{x_j^2 + y_j^2}$. Therefore,

$$\begin{aligned} \tilde{A} &= \frac{8 \int_0^{\frac{\pi}{4}} \int_0^{\frac{N_x}{2\cos\beta}} 2\gamma^2 \, d\gamma \, d\beta}{N_x N_x \pi} \\ &= \frac{2N_x^3}{3N_x^2\pi} \int_0^{\frac{\pi}{4}} \frac{1}{(\cos\beta)^3} \, d\beta \\ &= \frac{N_x}{3\pi}(\sqrt{2} + \ln(1+\sqrt{2})) \end{aligned} \tag{C.3}$$

Similar to Appendix A, Equation (C.3) must compensate for errors because $\tilde{A}$ must be a multiple of Δ_d. Therefore, a constant term is added to Equation (C.3) and $\tilde{A}$ is approximated to be:

$$\tilde{A} \approx \frac{N_x}{3\pi}(1 + \sqrt{2} + \ln(1+\sqrt{2})) \tag{C.4}$$

After measurements are copied from the sinogram space to a SVB, all voxel traces are flattened in the SVB with a much smaller amplitude (see Section 3.3.2 for more detail). To illustrate this idea, Figure C.2 shows a red voxel trace copied from the yellow sinusoidal sinusoidal band in the sinogram space to a SVB. After copying to the SVB, the red voxel trace has a smaller amplitude. To calculate the average voxel trace amplitude in the SVB, we can view the SV as an image, whose length and height are $N_{wh}\Delta_x$, where N_{wh} is the number of voxels a SV has on each side and Δ_x is the voxel width. Therefore, the average voxel trace amplitude in the SVB, denoted as $\tilde{A}_{SVB}$, can be computed by using Equation (C.4). After replacing N_x in Equation (C.4) with $N_{wh}\Delta_x$, $\tilde{A}_{SVB}$ can be computed as:

$$\tilde{A}_{SVB} \approx \frac{N_{wh}\Delta_x}{3\pi}(1 + \sqrt{2} + \ln(1+\sqrt{2})) \tag{C.5}$$

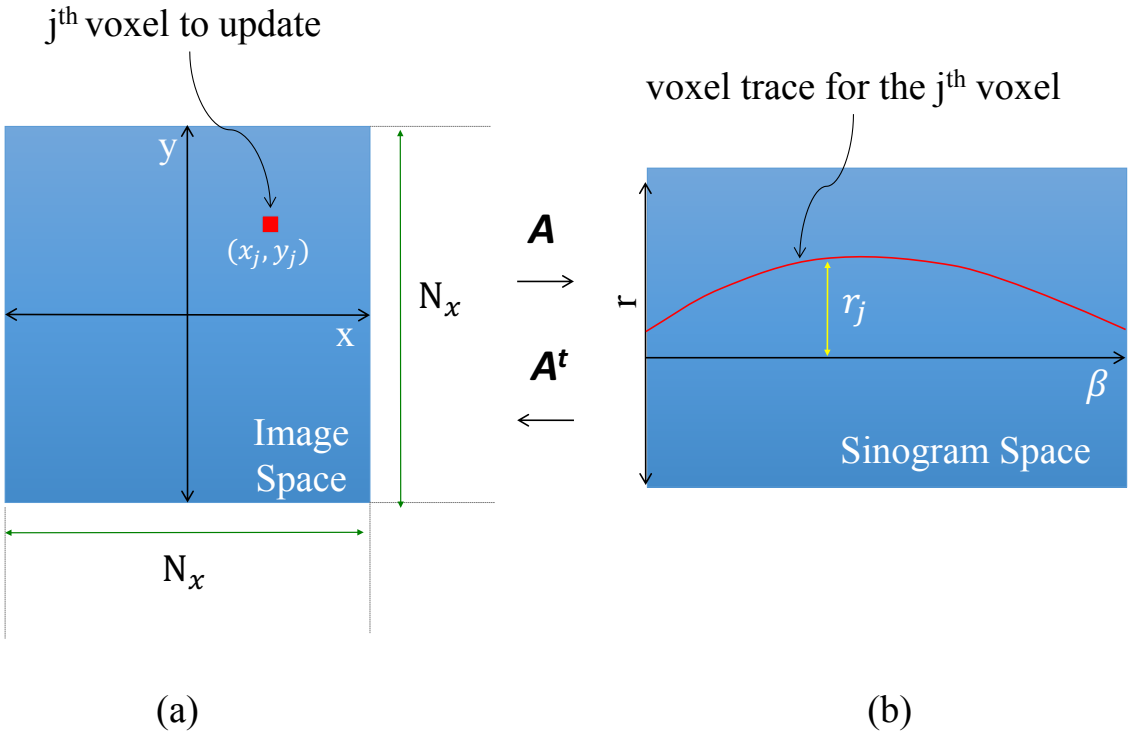


Fig. C.1.: (a) The red square in the image space is the j^{th} voxel, whose coordinate is (x_j, y_j). (b) The red trace is the measurements for the j^{th} voxel. The yellow bar r_j represents the voxel trace amplitude in the sinogram space.

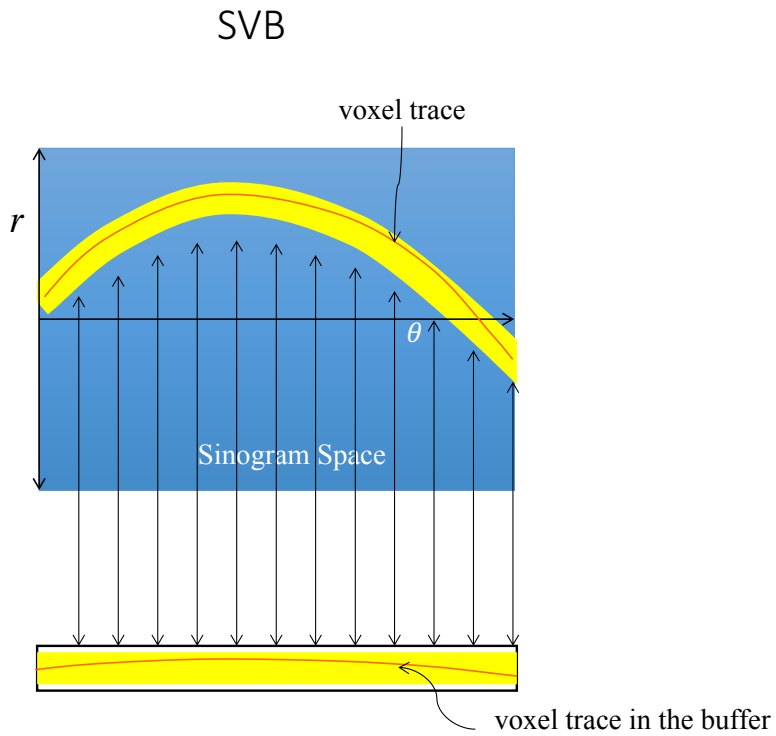


Fig. C.2.: After copying measurements from the sinogram space to SVB, the voxel trace becomes much flatter in the SVB with a much smaller amplitude.

D. THE VOXEL TRACE AVERAGE ABSOLUTE SLOPE

As explained in Appendix C, the voxel trace amplitude for j^{th} voxel, r_j, at view angle β is expressed as:

$$r_j(\beta) = y_j \cos\beta - x_j \sin\beta \tag{D.1}$$

Therefore, the slope of the voxel trace at view angle β is:

$$r_j^{'}(\beta) = -y_j \sin\beta - x_j \cos\beta \tag{D.2}$$

With Equation (D.2), the voxel trace average absolute slope in the sinogram space, denoted as $\tilde{S}$, can be computed in a way similar to Equation (C.3):

$$\begin{aligned}
\tilde{S} &= \frac{\int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_0^{\pi} |r^{'}(\beta)|\, d\beta\, dx_j\, dy_j}{N_x N_x \pi} \\
&= \frac{\int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_{\frac{-N_x}{2}}^{\frac{N_x}{2}} \int_0^{\pi} |-y_j \sin\beta - x_j \cos\beta|\, d\beta\, dx_j\, dy_j}{N_x N_x \pi} \\
&= \frac{N_x}{3\pi}(\sqrt{2} + \ln(1+\sqrt{2}))
\end{aligned} \tag{D.3}$$

We can notice that the average absolute slope equals to the average amplitude, $\tilde{A}$, computed in Equation (C.3). This result is not surprising because the analytical expressions for the amplitude (see Equation (D.1)) and the slope (see Equation (D.2)) are both sinusoidal functions with the same peak amplitude and period, but with different phase shift. Therefore, the analytical solutions for the average absolute slope and the average amplitude are the same.

To compensate for the quantization error, we add a constant term to Equation (D.3) to offset the error. Then, we can get:

$$\tilde{S} \approx \frac{N_x}{3\pi}(1 + \sqrt{2} + \ln(1+\sqrt{2})) \tag{D.4}$$

For the same rationale as in Equation (C.5), the voxel trace average absolute slope in the SVB, denoted as $m(\phi)$, can be computed as:

$$m(\phi) \approx \frac{N_{wh}\Delta_x}{3\pi}(1 + \sqrt{2} + \ln(1 + \sqrt{2})) \tag{D.5}$$

where N_{wh} is the number of voxels a SV has on each side and Δ_x is the voxel width.

VITA

VITA

Xiao Wang received B.A. degrees in Mathematics and Computer Science with honor from Saint John's University, MN, in 2012, and M.S. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2016. He is currently pursuing a PhD degree in electrical and computer engineering from Purdue University, under the supervision of Professor Charles Bouman and Samuel Midkiff. He is planning on graduating in August, 2017 and joins Harvard University as a postdoctoral research fellow.

Xiao Wang's research work focuses on applying high performance computing to imaging problems, especially image reconstruction on CT and MRI. He and his advisors are selected to be the 2017 ACM Gordon Bell Prize finalist for the research work presented in this dissertation.