High Performance Model Based Image Reconstruction

Xiao Wang
School of Electrical and Computer Engineering, Purdue University
wang1698@purdue.edu

Amit Sabne
School of Electrical and Computer Engineering, Purdue University
asabne@purdue.edu

Sherman Kisner
High Performance Imaging LLC
kisner@ecn.purdue.edu

Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
raghunathan@purdue.edu

Charles Bouman
School of Electrical and Computer Engineering, Purdue University
bouman@purdue.edu

Samuel Midkiff
School of Electrical and Computer Engineering, Purdue University
smidkiff@ecn.purdue.edu

Abstract

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 paper describes a new MBIR implementation that significantly reduces the computational cost of MBIR while retaining its benefits. It describes a novel organization of the scanner data into *super-voxels* (SV) that, combined with a *super-voxel buffer* (SVB), dramatically increase locality and prefetching, enable parallelism across SVs and lead to an average speedup of 187 on 20 cores.

Categories and Subject Descriptors I.4.5 [*Image Processing and Computer Vision*]: Reconstruction—Transform methods; D.1.3 [*Programming Techniques*]: Concurrent Programming—Parallel Programming

General Terms Applications, Algorithms

Keywords Multicore, Parallel algorithm, CT image reconstruction, MBIR

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. The classic example of a computational imaging system is X-ray tomographic imaging in which a volume is reconstructed from a set of projections.

Image reconstruction techniques generally fall into two categories: direct methods, such as filtered back projection (FBP), and iterative methods. Among iterative methods, Model Based Iterative Reconstruction (MBIR) [20, 22] has been shown to result in higher quality images in applications including explosive detection systems (EDS) [4, 5, 13, 17], medical imaging [20, 22], scientific and materials imaging [14]. MBIR can also work effectively with a wider range of scanner geometries. MBIR, however, suffers from a high computational cost. In typical applications, iterative reconstruction methods may require a factor of 10 to over 100 times the computation of FBP.

Broadly speaking, there are two approaches to MBIR: simultaneous methods [3, 4, 8, 21] and iterative coordinate descent (ICD) methods [18, 20, 26]. Simultaneous methods work by iteratively projecting an entire volume to be reconstructed into the measurement or *sinogram space*.[1] Simultaneous methods, however, have a number of disadvantages. To speed up convergence, these algorithms must be used with preconditioning methods [1, 8, 16], which must be custom designed for each CT imaging system and geometry.

As an alternative to these approaches, ICD has 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 ICD is a greedy optimization approach and each *voxel* (a single data point in the final image being reconstructed) is updated to best minimize the cost function, so there is tight feedback in the optimization process. In practice, ICD has been shown to converge in 3 to 6 iterations instead of the hundreds of iterations needed by simultaneous methods [4, 18]. While ICD has rapid convergence, currently it is believed that ICD requires operations that are more difficult to parallelize [10, 27]. Moreover, ICD exhibits poor cache locality because of the data layout which, as shown in Sec. 2.1, requires a 2D array of memory to be accessed in sinusoidal patterns [7, 23].

Research in high performance CT image reconstruction can be summarized as addressing two challenges: (1) increasing the amount of parallelism, and (2) enhancing the locality of refer-

PPoPP '16, March 12-16, 2016, Barcelona, Spain.

http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2851141.2851163

[1] The measurement or sinogram space stores all measurements collected by CT scanner system. It is called the sinogram space because the measurement of a voxel traces a sinusoidal pattern in the memory.

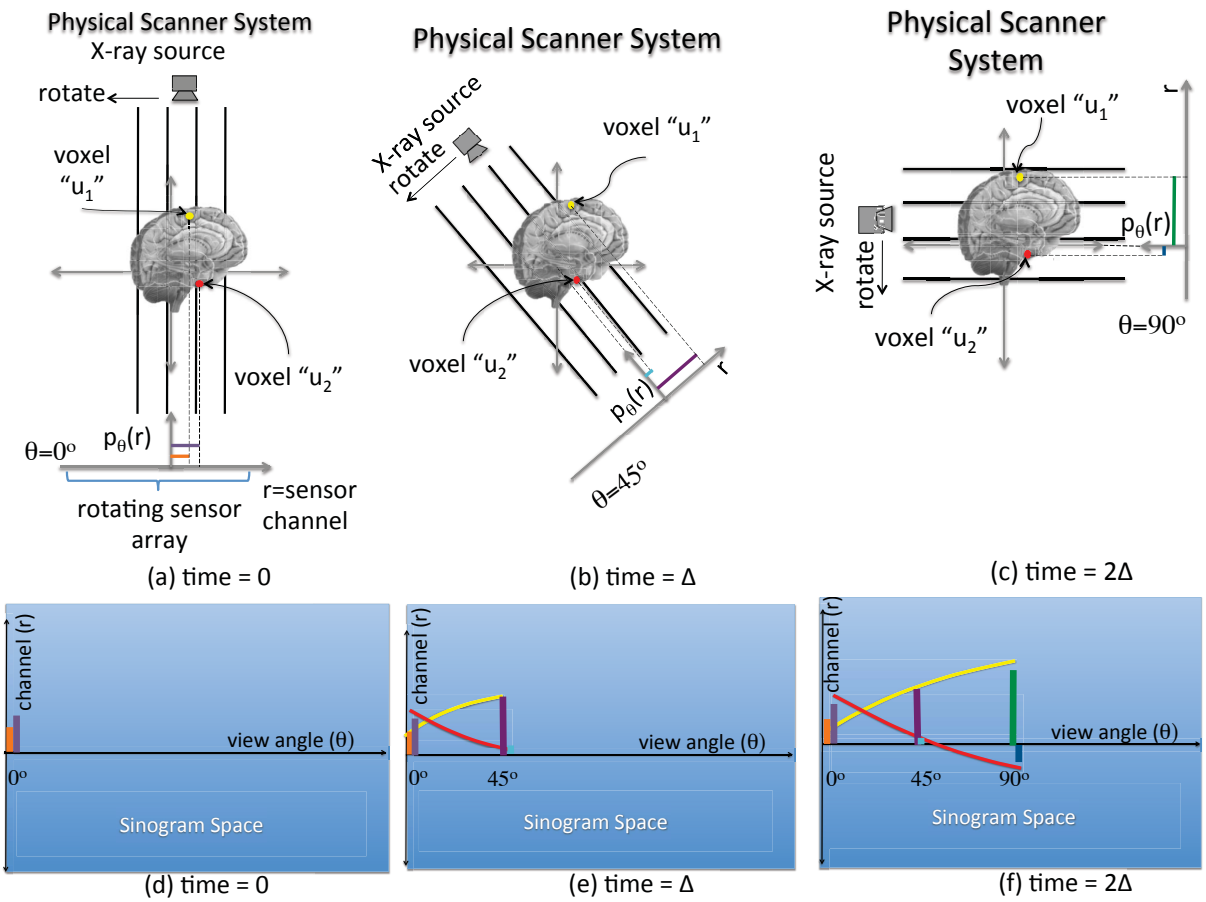


Figure 1: (a), (b) and (c) illustrate CT tomography system operations for voxels "u_1" and "u_2" when $\theta = 0°, 45°$and $90°$respectively. (d), (e) and (f) show how the measurement data collected from the CT scanner is organized into the sinogram.

ence. The ICD algorithm presents significant 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. Methods have been developed to find "loosely coupled voxels" sharing little or no data in common [19, 26]. By using loosely coupled voxels, it is shown in [10, 27] that, from an algorithmic perspective, updates of voxels could be done simultaneously with little effect on convergence rate. For example, in Table I of [10], parallelizing loosely coupled voxels leads to a convergence rate about 18% slower and a speedup of 2.3 times on 16 cores. 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. 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 nearby voxels that are strongly coupled and share more data, thus leading to better cache locality. Strongly coupled voxels, however, 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 enable good parallel executions.

This paper makes the following contributions:

1. It describes the performance issues inherent in MBIR;
2. It introduces and describes the ideas of the *super-voxel* (SV) and *super-voxel buffer* (SVB) to increase locality and prefetching;
3. It identifies and discuss available parallelism in MBIR;
4. It proposes a parallel super-voxel ICD algorithm (PSV-ICD) that simultaneously addresses the challenges of parallelism and locality;
5. It produces experimental results showing that PSV-ICD gives an average 14 speedup on a single core and an average 187 speedup on 2 Intel Xeon E5 processors with 20 cores in total.

The remainder of the paper is organized as follows. In Section 2, we review background information about CT and MBIR. In Section 3, we introduce the SV algorithm (SV-ICD/SVB) and its exploitation of cache locality and hardware prefetching. In Section 4, we investigate different levels of parallelism in high performance imaging, and introduce the PSV-ICD algorithm. In Section 5, we provide an evaluation of SV-ICD/SVB and PSV-ICD using datasets from an Imatron C300 CT Scanner, described in [5], 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 6, we present a discussion of the broader application of these techniques to other problems. Finally, we present our conclusions in Section 7.

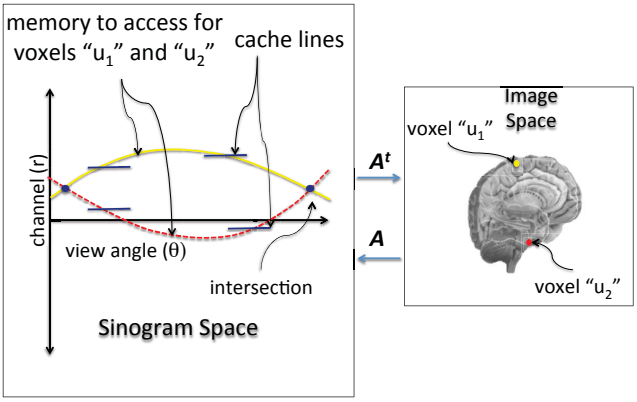


Figure 2: The sub-figure on the left shows the sinogram space storing all of the CT measurements. The sub-figure on the right shows the final reconstructed image. Notice that each single voxel in the image domain traces out a sinusoidal pattern in the sinogram domain. The sinusoidal pattern for voxel "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.

2. BACKGROUND

2.1 Computed Tomography

Figures 1 (a), (b) and (c) illustrate the basic concepts of how a typical parallel beam X-ray CT system operates. 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, θ. 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 $p_\theta(r)$, 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. Figures 1 (a), (b) and (c) also show how a CT system takes measurements for the voxels "u_1" and "u_2" when $\theta = 0°$, $45°$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 Figures 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 θ. 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 we connect channels r at different views θ 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 the voxels' corresponding measurement values in the sinogram following these sine wave patterns.

Modern processors access main memory by first transferring blocks of 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. 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 words in a cache line do 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. 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) [10, 23, 27], the most commonly known method for parallelizing ICD, has been demonstrated to only achieve a speedup of 2.3 on 16 cores.

2.2 Conventional ICD Algorithm

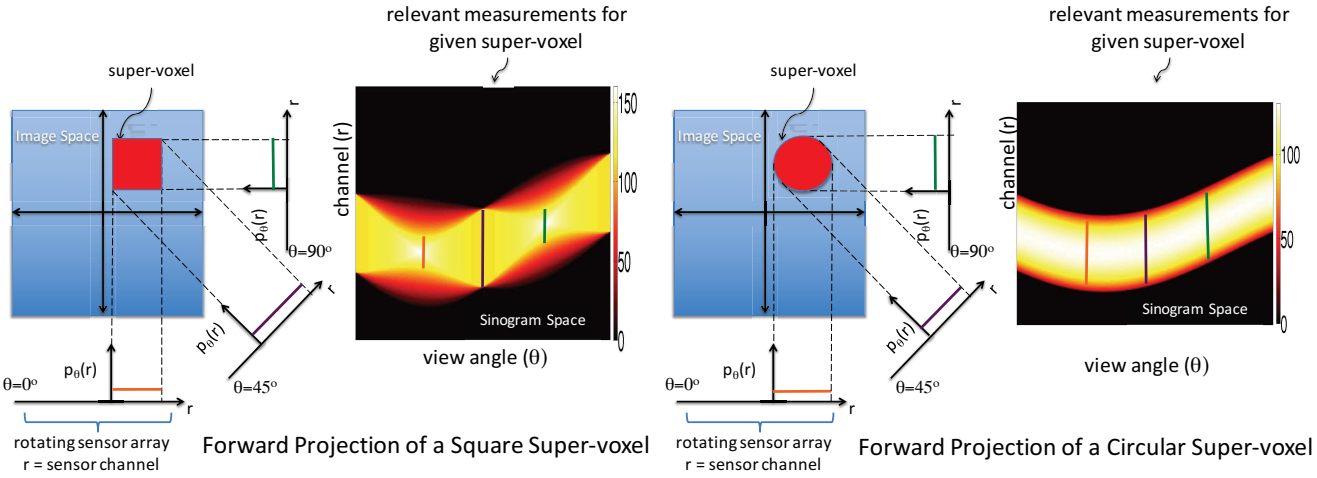
To better understand the key novelties of this paper, the central mathematical and algorithmic concepts of MBIR must first be briefly reviewed. MBIR is based on the numerical solution of an optimization problem described by

$$\hat{u} = \underset{u \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(v - Au)^T D(v - Au) + S(u) \right\} \qquad (1)$$

We consider the image u as a N dimension vector whose elements are called voxels and the data v as a vector of size M containing the CT measurement data. To relate this information with Figure 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 $S(u)$ 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 pre-computed 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 v_i. In the rest of this paper, diagonal entry $D_{i,i}$ is abbreviated as d_i. The ICD algorithm for solving equation (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 selected voxel u_j is given by

$$\hat{u}_j = \underset{u_j \geq 0}{\arg\min} \left\{ \frac{1}{2}(v - Au)^T D(v - Au) + S(u) \right\} \qquad (2)$$

In practice, the computation associated with the term $S(u)$ 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. Direct implementation of the ICD algorithm using the above formula requires the evaluation of the term $v - Au$ with each voxel update. Fortunately, there is a simple method for speeding up the computation by keeping a state variable to store this error term $e = v - Au$. In order to update the voxel u_j, however, it is necessary to evaluate terms involving the corresponding j^{th} column of the matrix A. More specifically, in order to compute an ICD update of voxel u_j, it is necessary to compute the first two derivatives



(a) (b)

Figure 3: (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.

of the cost function. The expressions for these two derivatives, denoted by θ_1 and θ_2, are given by

$$\theta_1 = -\sum_{i=1}^{M} d_i A_{ij} e_i \tag{3}$$

$$\theta_2 = \sum_{i=1}^{M} d_i A_{ij}^2 \tag{4}$$

where e_i is the i^{th} element in the error term. One can then write the minimization of the 1-D objective function for u_j explicitly as

$$u_j \leftarrow \underset{r \geq 0}{\operatorname{argmin}} \left\{ \theta_1 r + \frac{\theta_2 (r - \tilde{u}_j)^2}{2} + f(r, u_{\partial j}) \right\}, \tag{5}$$

$\tilde{u}_j$ is the j^{th} voxel's value before the update, and $f(r, u_{\partial j})$ is some function of the eight neighbors of the voxel u_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 [26].

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

1: $\tilde{u}_j \leftarrow u_j$
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: $u_j \leftarrow \operatorname{argmin}_{r \geq 0} \left\{ \theta_1 r + \frac{\theta_2 (r - \tilde{u}_j)^2}{2} + f(r, u_{\partial j}) \right\}$
5: $e \leftarrow e + A_{*j}(u_j - \tilde{u}_j)$

Algorithm 1 shows the pseudo code for the voxel update algorithm from [26]. The first step is to copy the voxel value to a local register $\tilde{u}_j$. We compute θ_1 and θ_2 using the formulas in Equation 3 and 4. Then we compute the voxel's updated value u_j efficiently by solving a 1-D optimization problem [26] in the Equation 5. Finally, we update the error term by forward projecting the update step $u_j - \tilde{u}_j$. When implemented properly [26], the computation of Algorithm 1 is dominated by steps 2, 3, and 5. The remaining steps typically represent a negligible amount of additional computation. Each iteration of the conventional ICD algorithm updates a voxel exactly once. Nonetheless, 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 [2]. 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 [25]. When a voxel is chosen for update, the reconstruction edge mask is first checked and updates are calculated only for the masked voxels.

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 performance of the image reconstruction algorithm.

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 answer this question, the *forward projection* [12] of a set of voxels can be used to determine where in the sinogram space the corresponding data values lie. For example, Figure 3(a) shows the forward projection for a square SV. At each view angle θ, 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 that 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(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.[2] Circular SVs, however, do not tessellate the

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

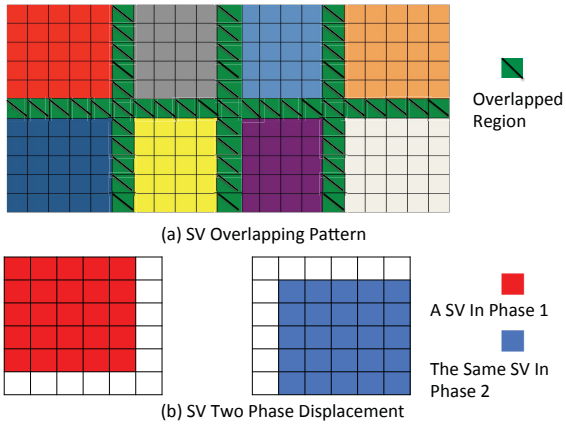


Figure 4: The layout and the overlaps of SVs in the image space using the two-phase mechanism. In (a), shaded green cells represent overlapping neighboring SVs. In (b), red SV is a phase 1 SV and the blue SV is the same SV in phase 2 but with a diagonal displacement.

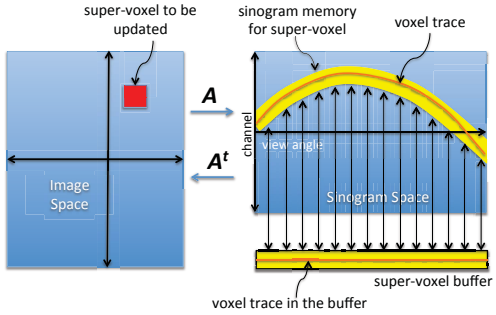


Figure 5: Memory access in sinogram space for SV updates. Processor hardware is used to load sinogram memory into a SVB. Note that a voxel trace is straightened out in the SVB.

image space. Consequently, circular SVs must have a large amount of overlap to fully cover the image space, leading to excessive computations on voxels in multiple overlapping SVs. At the same time, SVs of a different shape, which tessellate the image space with no overlap, will also lead to excessive computation because the algorithmic convergence could be adversely affected at SV boundaries.

To overcome these flaws, we use square SVs and a two phase convergence mechanism. With square SVs, data reuse is less compared to circular SVs, as can be seen in the forward projection for the square SV in Figure 3(a). Nonetheless, having each voxel belong to a single SV and eliminating redundant computations more than compensate for this disadvantage. Figure 4 illustrates the SV layout used to achieve the two phase convergence. In Figure 4(a), each phase of SVs of side length 6 provides a full covering of the image space while adjacent SVs overlap at the boundaries. This introduces limited redundant computations but it is compensated for by having faster algorithmic convergence. In Figure 4(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 9 of the Experimental Results Section 5.

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 will 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 for the SV is in a sinusoidal band with access patterns that are not recognized by the prefetcher. Figure 5 graphically illustrates this band pattern in which a red block represents the SV to be updated in 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 5 are copied into the SVB shown in the same figure. The SVB lays out the SV data so that memory accesses follow a "straight line" pattern that is ideal 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.

Theoretical analysis of SVB access patterns indicates that the average number of consecutive memory locations accessed in the SVB for a single voxel update along the view direction, denoted Run_{SVB}, is given by

$$Run_{SVB} \approx 2N_v/R, \tag{6}$$

where N_v is the number of views and R is the side length of the square SV. For images with square SVs with 13 voxels along each side and 720 views, it can be shown analytically that the average run length of data is 110. 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 given in Section 5, showing an average tenfold increase of the prefetch hit rate.

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.

4. Improving parallel execution

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

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 θ 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 $u_j \in a$ **do**
10: Voxel Update(j, u, SVB_e, SVB_d)
11: **end for**
12: **end for**
13: **for** each view θ 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**

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, 19, 24, 26]. 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) and (4) in Section 2.2 can be considered to be an example of intra-voxel parallelism since it parallelizes multiple data operations in a single voxel update. Moreover, GCD [10] 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 multicore 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 u_j in the $voxelSet$ **do in parallel**
2: $\tilde{u_j} \leftarrow u_j$
3: $\theta_1 \leftarrow Compute$ as in Equation (3)
4: $\theta_2 \leftarrow Compute$ as in Equation (4)
5: $u_j \leftarrow \text{argmin}_{r \geq 0}\{\theta_1 r + \frac{\theta_2 (r - \tilde{u_j})^2}{2} + f(r, u_{\partial j})\}$
6: lock
7: $e \leftarrow e + A_{*j}(u_j - \tilde{u_j})$
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 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. 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 the 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 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 [10]) 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, 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 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.

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 $u_j \in sv^k$ 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**

The augmented SVB, illustrated in Figure 6, 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. Shown in Figure 8 (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 3.1, resulting in convergences as fast as the sequential algorithm, as seen in Figure 9 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 6. 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 $SVB_{\Delta e}^k$, is initialized to 0 and all changes of data relative to k^{th} SV are made to this buffer only. When SV k has been updated, the sinogram data changes kept in $SVB_{\Delta e}^k$ 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 does not happen. The lock waiting overhead exists only after all voxels within a SV are updated. In Algorithm 4, $SVB_{\Delta e}^k$ is created in step 8 after all voxels in the SV are updated and it represents the additions of data made to this buffer. At the end of the algorithm in step 10, these additions of data are atomically added back to the memory.

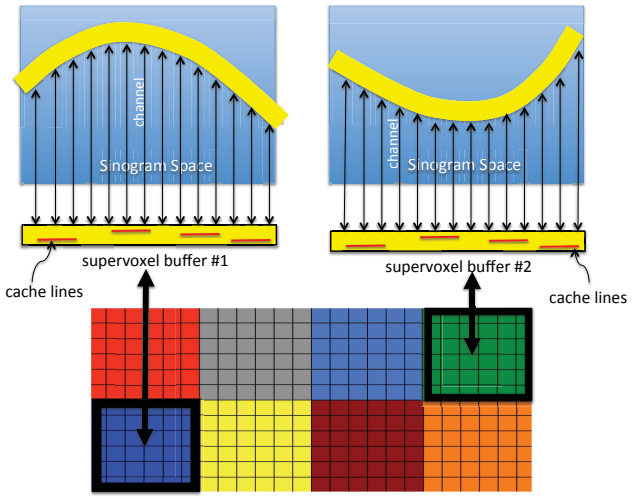


Figure 6: Parallel SVs update with separate augmented SVBs. Two different SVs have different but overlapping sinogram bands.

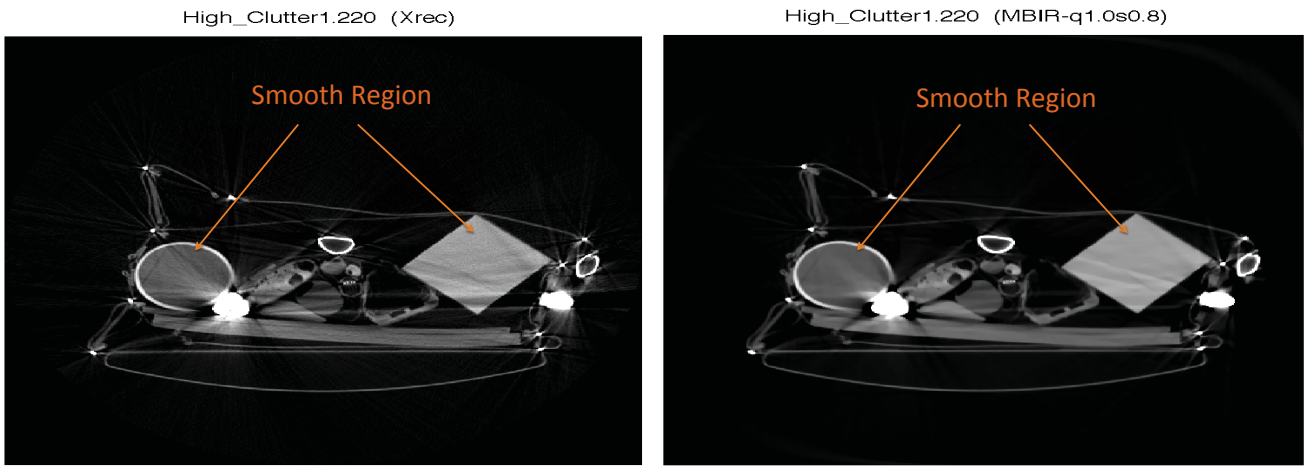
5. Experiment Results

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 [5]. 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); (4) 512×512 reconstructed image size with an embedded circular ROI.

Figure 7 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 1 lists detailed results. Because the baseline ICD, SV-ICD/SVB and PSV-ICD do not update all voxels (see Section 2 and [26]) 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[3] between the result and a fully converged reconstruction after approximately 20 equits. In extensive experimentation 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 been implemented in OpenMP using the Intel C++ Compiler version 15.0.3.187 and

[3] 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)

Figure 7: One reconstructed slice example in Imatron C-300 scans by FBP and MBIR. The image on the left is this slice fully converged by FBP. The image on the right is done by MBIR. Note the image quality enhancement in the smooth region.

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 2.2 with 1 core. The baseline code is accessible at https://engineering.purdue.edu/ bouman/software/tomography/mbirct/ [15] and is used by researchers in many real-world applications, for example, [13, 14, 19]. 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 4, inter-slice parallelism has been widely used with good parallel performance [19]. 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.

$$T_r = \frac{N_F N_e}{O_F E_F} \tag{7}$$

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 seconds given 100% GFLOP efficiency.

Using this framework, Table 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 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 8(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 with the number of cores increases. Figure 8(b) shows L1, L2 and L3 cache level data cache miss rate of PSV-ICD at different number of cores. With more cores and more memory copies, 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.

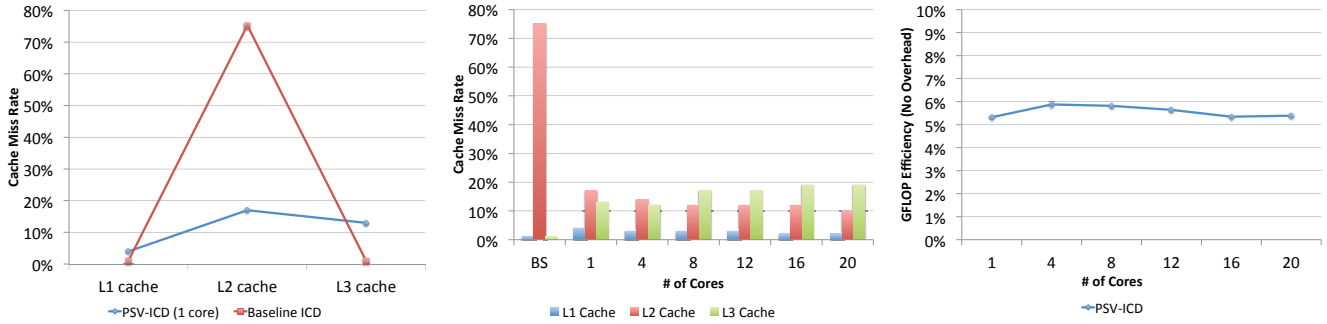
In addition, the L2 cache prefetching also contributes to the L2 cache miss rate reduction. Baseline ICD has 0.14% prefetch rate, 3.6% prefetch hit rate for its L2 cache prefetcher on average. PSV-ICD (20 cores) L2 prefetch rate increases to 0.94% and its prefetch hit rate is 33.6% on average. We can 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 microarchitecture has no event type that can directly measure them.

These reductions in the data cache miss and prefetch hit rate increases have a direct effect on E_F, the processor floating point operations efficiency. If we compare column 2 and column 3 in Table 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 multicore communication cost and waiting on locks. For serial Baseline ICD, there is no multicore 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 8(c) shows E_F at different numbers of cores when the multicore 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 8(d). The important result of

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

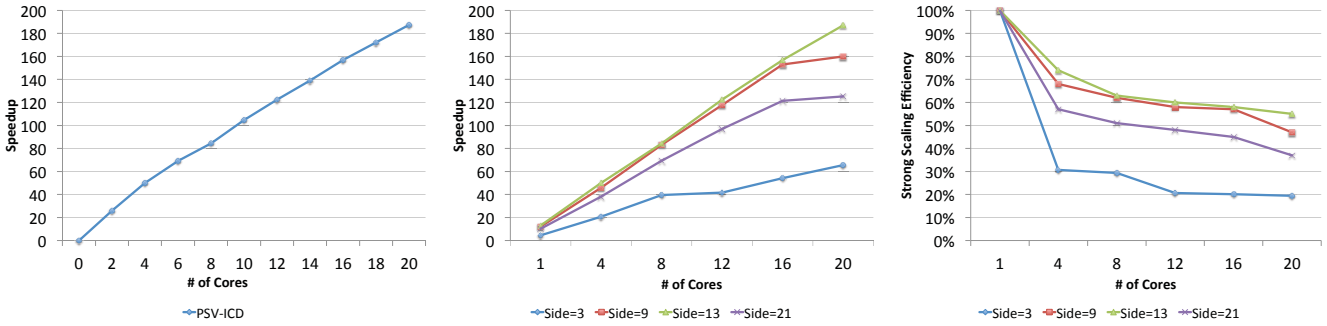
Table 1: Table of performance measurement for all algorithms discussed in this paper. Column 2 is the baseline ICD code. Column 3 is SV-ICD/SVB. Columns 4-6 are PSV-ICD with 4 cores, 16 cores and 20 cores respectively. Note that row 6, T_r, is the average actual reconstruction time and row 8 is the speedup of T_r at 20 cores comparing with the sequential baseline ICD.



(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.

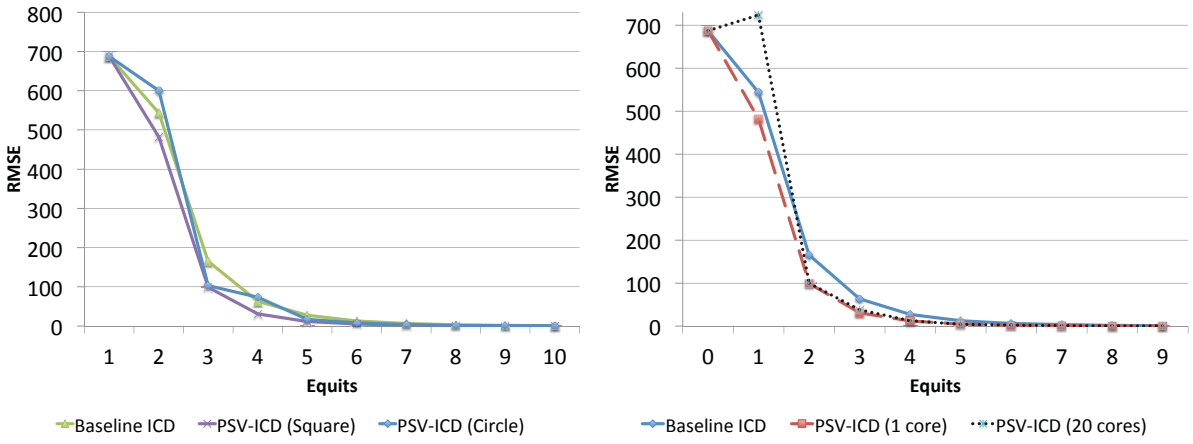
(e) PSV-ICD strong scaling speedup at different square SV side lengths.

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

Figure 8

this figure is that the speedup is almost linear in the number of cores. This shows that PSV-ICD and 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. In various experiments on PSV-ICD speedup, SV size has a significant fundamental impact on the speedup. Figure 8(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 size of 81 voxels in the SV for example. From Figure 8(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 pressures and reduce the cache hit rate. At the same time, a too small SV size increases parallel communications frequency and lock waiting overhead in step 10 of Algorithm 4. VTune analyzer shows that PSV-ICD with 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 8(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 for 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 SV side length of 21 reduces the lock wait time but has a much steeper parallel efficiency drop compared with SV side length of 13 because of the reduced cache hit rate from



(a) (b)

Figure 9: (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.

the large SV size. In addition, Figure 8(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. Our results indicate that our PSV-ICD can converge to a RMSE of 8 HU after 4.1 equits on average on 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 9(a) shows a plot of RMSE versus equits for PSV-ICD (1 core). For the two phase convergence 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 9(b) shows the convergence speed of baseline ICD and PSV-ICD with different cores. Note that PSV-ICD in Figure 9(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 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.

6. The Broader Application to Other Domains

While this paper focuses on the problem of high-performance CT systems, the methodology we describe is applicable to a broad range of sensing problems that can be expressed in the form

$$\hat{x} = \arg\min_{x} \{\|y - Ax\|_{\Lambda} + S(x)\} \tag{8}$$

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 (8). 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,[4] the data is simply a multidimensional array of quantitative measurements of the environment 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 8, 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 [6].

For the general problem of equation (8), the PSV-ICD algorithm provides a parallel framework for computing the compressive sensing problems efficiently. The traditional approach has been to find columns of A that are uncorrelated [11]. More specifically, if we define the correlation between columns to be

$$cor(i,j) = \sum_{k=1}^{N} |A_{k,i}| \cdot |A_{k,j}| \tag{9}$$

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 [10, 27]. Columns that

[4] 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.

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 this paper, 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 (8), 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 4.

7. Conclusion

While MBIR provides high quality image reconstructions and is agnostic to scanner geometries, it has been considered impractical because of its long running 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 also note that we are still utilizing less than 6% of processor efficiency despite significant performance gains. Thus, future work 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.

Acknowledgments

We would like to thank John Reppy for his helpful comments. In addition, this research was supported 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 authors 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.

References

[1] S. Basu and Y. Bresler. $O(N^2 \log_2 N)$ Filtered Backprojection Reconstruction Algorithm for Tomography. *IEEE Transactions on Image Processing*, 9(10), 2000.

[2] J. E. Bowsher, M. Smith, J. Peter, and R. J. Jaszczak. A Comparison of OSEM and ICD for Iterative Reconstruction of SPECT Brain Images. *Journal of Nuclear Medicine*, 79(5), 1998.

[3] N. 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*, 12(1), 1993.

[4] 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*, volume 9401, 2015.

[5] 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.

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

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

[8] J. Fessler and S. D. Booth. Conjugate-Gradient Preconditioning Methods for Shift-variant PET Image Reconstruction. *IEEE Transactions on Image Processing*, 8(5), 1999.

[9] 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.

[10] J. A. Fessler, E. Ficaro, N. Clinthorne, and K. Lange. Grouped-Coordinate Ascent Algorithms for Penalized-Likelihood Transmission Image Reconstruction. *IEEE Transactions on Medical Imaging*, 16(2), 1997.

[11] 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.

[12] C. Hoilund. The Radon Transformation. `http://mlsp.cs.cmu.edu/courses/fall2012/lectures/Carsten_Hoilund_Radon.pdf`, 2007.

[13] 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.

[14] 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)*, pages 1–5, 2013.

[15] P. Jin, S. J. Kisner, T. Frese, and C. A. Bouman. Model-Based Iterative Reconstruction (MBIR) Software for X-ray CT. Available from https://engineering.purdue.edu/ bouman/software/tomography/mbirct/, November 2013.

[16] C. Kamphuis and F. J. Beekman. Accelerated Iterative Transmission CT Reconstruction Using an Ordered Subsets Convex Algorithm. *IEEE Transactions on Medical Imaging*, 17(6), 1998.

[17] 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*, pages 444–447, June 2012.

[18] 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*, volume 5, pages 2708–2710, 2005.

[19] 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*, 1(2):96–111, June 2015.

[20] K. Sauer and C. Bouman. A Local Update Strategy for Iterative Reconstruction from Projections. *IEEE Transactions on Signal Processing*, 41(2), 1993.

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

[22] 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*, 34(11), 2007.

[23] X. Wang, C. A. Bouman, and S. P. Midkiff. High Performance Model Based Image Reconstruction. In *2015 ACM/IEEE Conference on Supercomputing*, November 2015. URL `http://sc15.supercomputing.org/sites/all/themes/SC15images/src_poster/poster_files/spost107s2-file1.pdf`.

[24] 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.

[25] 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.

[26] 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*, 20(1), 2011.

[27] J. Zheng, S. S. Saquib, K. Sauer, and C. A. Bouman. Parallelizable Bayesian Tomography Algorithms with Rapid, Guaranteed Convergence. *IEEE Transactions on Image Processing*, 9(10), 2000.