Massively Parallel 3D Image Reconstruction

Xiao Wang
Purdue University

Amit Sabne
Microsoft Corporation

Putt Sakdhnagool
Purdue University

Sherman J. Kisner
High Performance Imaging LLC

Charles A. Bouman
Purdue University

Samuel P. Midkiff
Purdue University

ABSTRACT

Computed Tomographic (CT) image reconstruction is an important technique used in a wide range of applications. Among reconstruction methods, Model-Based Iterative Reconstruction (MBIR) is known to produce much higher quality CT images; however, the high computational requirements of MBIR greatly restrict their application. Currently, MBIR speed is primarily limited by irregular data access patterns, the difficulty of effective parallelization, and slow algorithmic convergence.

This paper presents a new algorithm for MBIR, the *Non-Uniform Parallel Super-Voxel* (NU-PSV) algorithm, that regularizes the data access pattern, enables massive parallelism, and ensures fast convergence. We compare the NU-PSV algorithm with two state-of-the-art implementations on a 69632-core distributed system. Results indicate that the NU-PSV algorithm has an average speedup of 1665 compared to the fastest state-of-the-art implementations.

1 JUSTIFICATION

Significant advances in the design and implementation of a high-performing, scalable 3D Model-Based Iterative Reconstruction (MBIR) are presented. These advances give an algorithm that scales to 69632 cores on the Cori Xeon-Phi Knights Landing cluster and is 1665 times faster than the state-of-the-art.

Attributes	Contents
Category	*Time-to-solution, scalability*
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*

2 INTRODUCTION

Computed Tomography (CT) is a widely-used imaging technique for the reconstruction of 3D volumes. Applications for these reconstructions include security baggage scanning [8, 9, 18, 19, 30], medical and biological imaging [26, 27], and scientific and materials imaging [13, 17, 21, 29].

SC17, Denver, CO, USA
© 2017 ACM. 978-1-4503-5114-0/17/11...$15.00
DOI: 10.1145/3126908.3126911

MBIR [26, 27] is a regularized 3D reconstruction method, based on Bayesian estimation. Extensive prior research has shown that MBIR provides higher image quality than other methods [21, 26, 27]. In addition, MBIR requires only one fifth of the typical X-ray dose, and thus greatly reduces data acquisition time [2]. Because of these benefits, MBIR has great potential to be adopted in next-generation imaging systems [16]. However, MBIR's improved image quality requires several orders of magnitude more computation compared to traditional methods [30, 33]. Therefore, MBIR is considered impractical for many applications [2, 16].

Algorithms for MBIR can be summarized into two categories: global update and local update methods, with each having advantages and disadvantages. For global update methods [7, 10], all 3D-pixels in a volume, also known as *voxels*, are updated simultaneously in every iteration. Therefore, global update methods are naturally parallel and enable good scalability. However, massive parallelism also leads to slow algorithmic convergence [26, 30], sometimes requiring hundreds of iterations [8, 20].

In contrast, local update methods, such as coordinate descent [3, 4, 11, 22, 26, 31, 34] have much faster convergence with fewer iterations [33] and allow for the simultaneous update of small groups of voxels [11, 25, 34]. There are two state-of-the-art implementations for local update methods: Time-Interlaced MBIR (TIMBIR)[1] [13, 21] and Parallel Super-Voxel Iterative Coordinate Descent (PSV-ICD) [24, 30], 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 synchronization overhead among cores. However, this approach also reduces cache locality and limits scalability. Alternatively, PSV-ICD updates groups of spatially close voxels that share much data. In this case, a single core that sequentially updates this group of spatially close voxels can reuse data, which benefits cache locality. Multiple cores can then update spatially separated groups of voxels to reduce synchronization overhead (or atomic operation overhead). PSV-ICD, however, is a single node implementation and cannot scale. Therefore, existing implementations for local update methods require a 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 a 2D array, following an irregular pattern in memory. Importantly, this access pattern is different for each voxel, so there is no single reorganization of the data can resolve this SIMD inefficiency.

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

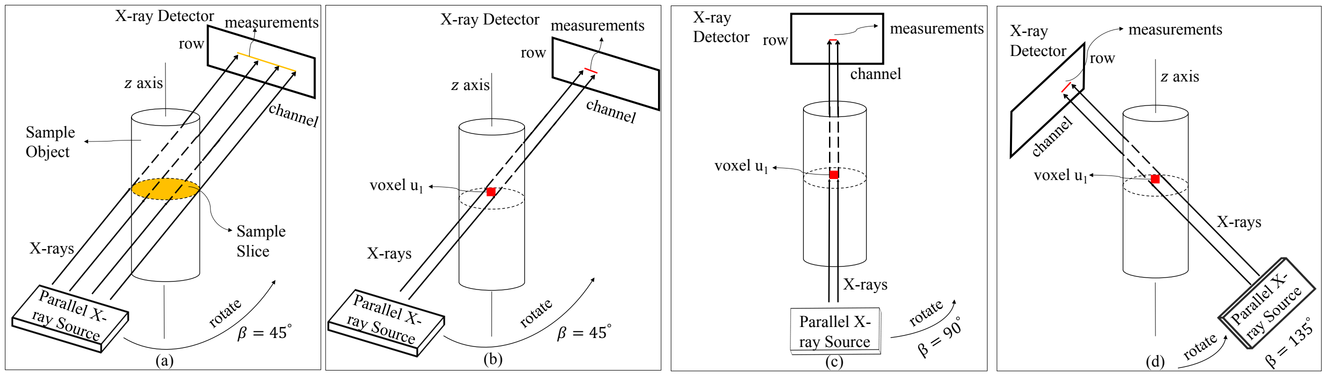


Figure 1: (a) A CT system setup. (b) At view 45°, X-rays pass through voxel u_1, making a projection on the left side of the detector. (c) At view 90°, the projection for voxel u_1 shifts to the center of the detector. Note that the projection at view 90° is shorter than at view 45°. (d) At view 135°, the projection shifts to the right side of the detector.

In this paper, we present a massively parallel algorithm, *Non-Uniform Parallel Super-Voxel* (NU-PSV), that directly addresses the four major performance issues of MBIR: (1) data access pattern regularity, (2) cache locality, (3) parallelism, and (4) algorithmic convergence. The NU-PSV algorithm utilizes a *Three-Dimensional Super-Voxel* (3DSV) to ensure good cache locality and low parallel overhead while retaining the fast convergence of a local update method. It also incorporates a *Block Transposed Buffer* (BTB) data layout to completely regularize the memory accesses and enable SIMD operations.

In summary, in this paper we:

- Introduce the concept of the 3DSV for 3D image reconstruction;
- Describe the BTB design that regularizes data access and improves SIMD operations;
- Describe the NU-PSV algorithm and provide experimental results showing that NU-PSV scales to 69632 cores with a speedup of 1665 compared to PSV-ICD and 9776 compared to TIMBIR.

In Section 3, we review background information about CT scanner systems and the state-of-the-art implementations. In Section 4, we introduce the 3DSV design and the NU-PSV algorithm. In Section 5, we present three reconstruction performance evaluations for NU-PSV on different datasets. In Section 6, we discuss how NU-PSV can be applied to compressive sensing problems in general. Finally, we present our conclusions in Section 7.

3 BACKGROUND

This section describes a generic system for CT, along with current state-of-the-art implementations for MBIR.

3.1 3D Computed Tomography

Figure 1(a) illustrates a typical CT scanner system, consisting of an X-ray source and a detector mounted on opposite ends of a rotating gantry, with the X-ray detector elements along the vertical direction called *rows*, and the detector elements along the horizontal direction called *channels*. 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 images, or *views*, are taken by the detectors at discrete rotation angles, denoted by β in the figure. 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, the reconstruction is obtained in slices, where a slice is defined as a cross-section of the sample object found along a fixed value of z. Then, an inverse is computed from the measurements to determine the radio-density of each slice. 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 1(a), a sample slice within the sample object is shown as a shaded circle. Notice that its measurements project to the shaded horizontal line along the same row of the detector.

Figures 1(b), (c), and (d) illustrate how a single voxel in a slice maps to the detector measurements. Let u_1 denote a voxel in the sample slice, shown as a red square. Figure 1(b) shows the set of channels, represented as a red bar on the detector, that receive measurements from u_1 at view angle 45°. We can note that the set of channels are located on the left side of the detector. Figures 1(c) and (d) shows the set of channels that receive measurements from u_1 at view angles 90° and 135°, respectively. Note that the set of channels shift to the center and right side of the detector at view angles 90° and 135°, respectively.

To organize and process the measurements from different view angles, the measurements for a slice are organized into a view-by-channel 2D array, known as a *sinogram*. Figure 2(a) illustrates such a sinogram, with the red sinusoidal trace representing the measurements for voxel u_1. When u_1 is updated, its associated measurements must be accessed following this red sinusoidal trace in the memory. Unfortunately, no cache line can efficiently access measurements in memory following a sinusoidal trace, which explains why MBIR has poor cache locality. Furthermore, the amplitude and phase of the sinusoidal trace is different for each voxel, so no universal reordering of the measurements can regularize the memory access pattern for all voxels.

Another interesting feature is that the width of the voxel trace varies at different view angles. Figure 1(b), (c) and (d) illustrate

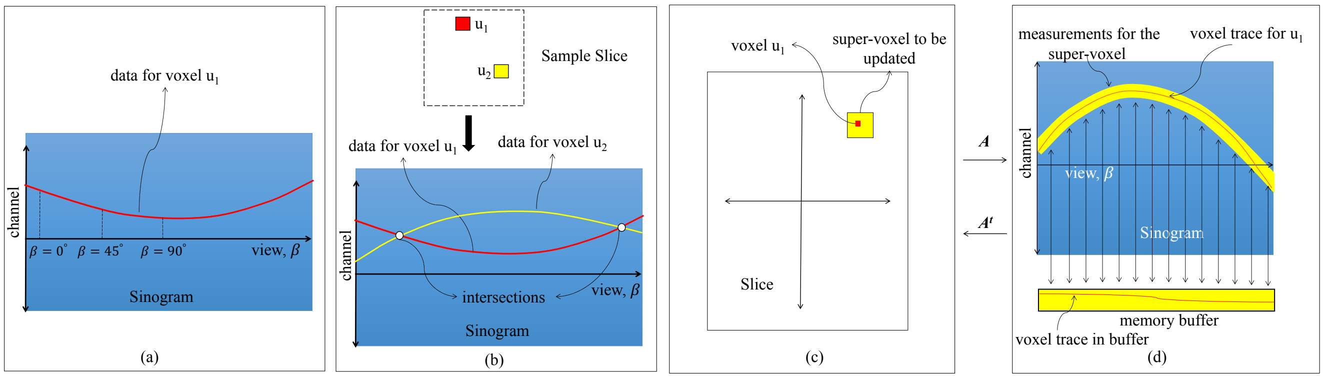


Figure 2: (a) The update of u_1 follows a sinusoidal trace in sinogram. (b) The sinusoidal traces for u_1 and u_2 intersect at white dots. (c) Shows a super-voxel in a slice. (d) Measurements for the super-voxel follow a sinusoidal band in sinogram and measurements for u_1 fall within the sinusoidal band. After copying measurements to a memory buffer, the voxel trace for u_1 still curves up and down with a smaller amplitude.

this effect. Note that the length of projection onto the detector at $\beta = 90°$, shown in Figure 1(c), is the shortest; but at angles $\beta = 45°$ and 135°, shown in Figure 1(b) and (d), the length of projection is across the diagonal of the square voxel and is the longest. For a cubic voxel specifically, the length of projection increases by a factor of $\sqrt{2}$ at angles $\beta = 45°$ and 135°. Consequently, the combination of sinusoidal memory access and varying voxel trace width leads to irregular data access in MBIR.

Figure 2(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 traces in the sinogram, respectively. Note that the voxel traces for u_1 and u_2 intersect at the white dots. Therefore, if u_1 and u_2 are updated in parallel, 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 [14]. Such voxel traces, however, also share few measurements and reduce cache locality. In contrast, updating groups of spatially close voxels in parallel leads to their voxel traces close to each other. Therefore, cache locality improves when spatially close voxels are updated, but the associated synchronization overhead for these updates is much worse.

3.2 State-of-The-Art

The 3D MBIR reconstruction is computed as the solution to the following optimization problem:[2]

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

where *argmin* is the minimizer that returns the reconstruction, $\hat{x}$; y is a vector of length M, whose elements are the measurements in the sinogram; and 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; and $R(x)$ is a regularizing function. The first term of the Equation (1), given by $\frac{1}{2}(y - Ax)^T \Lambda (y - Ax)$, is known as the data 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 [26].

To compute the solution to Equation (1), coordinate descent methods update small groups of voxels in parallel [11, 25, 34]. To update each voxel x_j, Algorithm 1 summarizes the key operations.

Algorithm 1 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: $\alpha \leftarrow \operatorname{argmin}_{\alpha \geq -x_j} \left\{ \theta_{1,j}\alpha + \frac{1}{2}\theta_{2,j}\alpha^2 + R(x_j + \alpha) \right\}$
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 the 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 neighbor voxels of x_j.[3] In Algorithm 1, steps 1 and 2 calculate the first and second derivatives required for step 3. In steps 3 and 4, α is chosen to minimize the cost function and x_j is updated, In step 5, e is updated so that it is consistent with the updated voxel value, x_j.

Next, we review the two state-of-the-art implementations for coordinate descent methods, TIMBIR and PSV-ICD.

TIMBIR For TIMBIR, each core updates all the voxels in a single slice, and different cores update different slices of the volume in

[2]In fact, Equation (1) represents the general form for most compressive sensing problems [24, 30]. Therefore, its solution is of general interest (see Section 6).

[3]8 nearest voxels in the same slice and 2 nearest voxels in the adjacent slices.

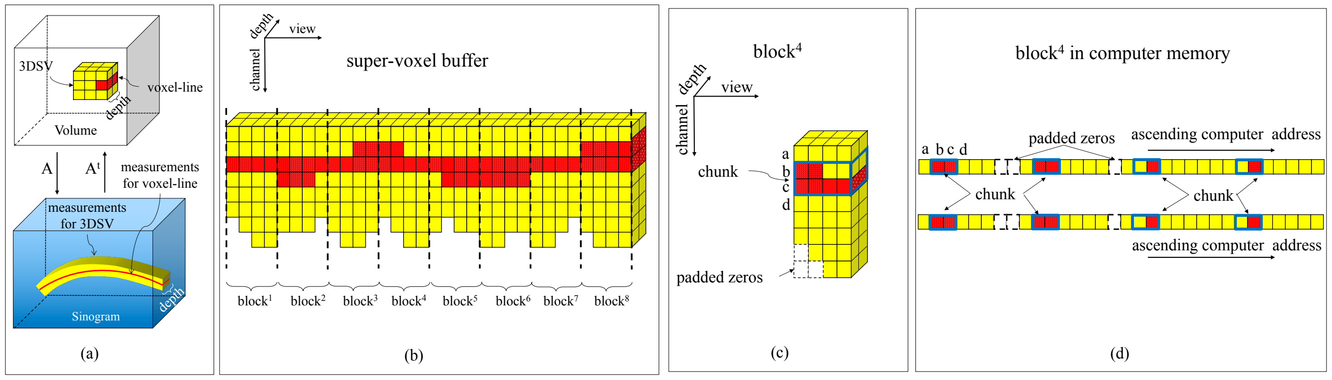


Figure 3: (a) Shows a 3DSV of size $3 \times 3 \times 2$. A voxel-line in the 3DSV is shaded in red. Measurements for the 3DSV follow a sinusoidal band in the sinogram. (b) A super-voxel buffer. Notice that measurements for the red voxel-line trace curves up and down in the super-voxel buffer with a small amplitude. (c) *block*^{4} of the super-voxel buffer with padded zeros and chunk design. (d) The memory layout for *block*^{4}. Notice that the measurements in the chunk are scattered in memory.

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 because the updates of adjacent slices have dependencies (see the prior function $R(x_j+\alpha)$ in Algorithm (1)). Therefore, TIMBIR can only update non-adjacent slices in parallel.

PSV-ICD 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 sequence. 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 2(c) shows an example of a super-voxel. The measurements of the super-voxel follow the yellow sinusoidal band in the sinogram, shown in Figure 2(d). Every voxel of the super-voxel, such as u_1, has a voxel trace inside the sinusoidal band. Therefore, voxels of a super-voxel 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, shown as a yellow rectangle in Figure 2(d). Then, every voxel trace in the memory buffer is flattened. Note that the voxel trace for u_1 becomes much more linear in the memory buffer, improving hardware prefetching. The voxel trace, however, still curves up and down in the memory buffer, albeit with a smaller amplitude.

Despite the benefits outlined above, PSV-ICD has two major disadvantages: 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 requires fine-grained atomic operations. Therefore, PSV-ICD is limited to a single shared memory node and is not capable of large-scale parallelism.

4 INNOVATIONS

This section presents the *Non-Uniform Parallel Super-Voxel* algorithm (NU-PSV). Section 4.1 discusses how NU-PSV regularizes data accesses and improves SIMD operations and prefetching. Section 4.2 discusses the parallelization of NU-PSV, and Section 4.3 discusses the improved algorithmic convergence.

4.1 Improving Data Access

We define a 3DSV as a group of voxels in the shape of a rectangular cuboid in the volume. Figure 3(a) shows a 3DSV of size $3 \times 3 \times 2$, where the width and height are 3 and the depth is 2. In addition, we define a sequence of voxels along the depth (z axis) of a 3DSV as a *voxel-line* [29, 33]. For the example in Figure 3(a), 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 3(a) also 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 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 shared measurements from the sinusoidal band, the measurements are repeatedly accessed as voxel-lines are updated. Therefore, the 3DSV design retains the cache locality benefit of the PSV-ICD algorithm.

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

Within the super-voxel buffer, all measurements for a voxel-line, shaded in red in Figure 3(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 and super-voxel buffer width (as in Figure 3(b)) still lead to irregular data access. To enable efficient SIMD operations, memory accesses must be regular and follow a pattern of $ik + j$, where i, j are sequences of contiguous integers and K is a constant

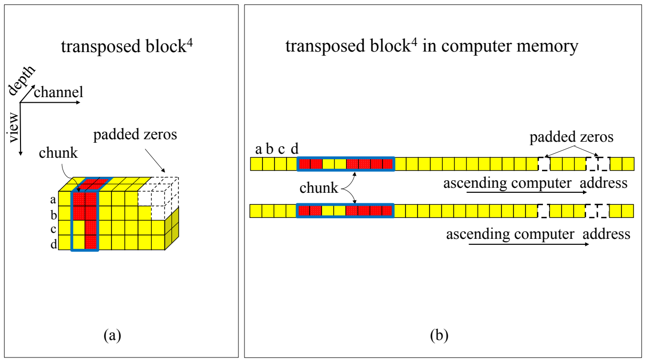

transposed block4
depth
channel
view
chunk
padded zeros
a
b
c
d
(a)
transposed block4 in computer memory
a b c d
padded zeros
chunk
ascending computer address
ascending computer address
(b)

Figure 4: (a) Shows the transposed *block*^{4}. (b) Shows the memory layout for the transposed *block*^{4}.

integer. 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 data structures. We define a *block* to be a fixed number of contiguous views within a super-voxel buffer, and denote the i^{th} block by $block^i$. Figure 3(b) shows a super-voxel buffer consisting of 8 blocks, with each block composed of 4 views. In addition, we define a *chunk* as a rectangular cuboid, circumscribing the sinusoidal voxel-line trace in a block. Figure 3(c) illustrates $block^4$ of the super-voxel buffer, and the chunk contained within $block^4$ is outlined with a bold blue line. To have a constant trace width, all the measurements in a chunk are accessed when processing a voxel line, even though only a portion of them are needed. For convenience, we call the measurements required for a voxel-line update *essential measurements*, shown in red in Figure 3(c). The unneeded measurements for the voxel-line update are called *redundant measurements*, shown in yellow in Figure 3(c).

In addition to the chunk design, each block is padded with zeros so that the buffer width, namely the number of channels of a block, is constant. Figure 3(c) illustrates this zero padding in which six zeros are added to the bottom of $block^4$, making $block^4$ a rectangular cuboid. Although redundant measurements and zero paddings moderately increase cache pressure by adding unneeded data, the performance gain from the regular data access far outweighs the loss from increased cache pressure (see experiments in Section 5.1).

Since the voxel trace and the buffer widths are both constant, measurements in a chunk have completely regular data access. Figure 3(d) shows the resulting memory layout for $block^4$. Since measurements are laid out along the channel direction in memory, measurements *a, b, c,* and *d* in Figure 3(c) have contiguous memory locations. Notice that the memory access pattern in Figure 3(d) is regular, with measurements in the chunk scattered into 8 groups and each group having two measurements.

While the memory access shown in Figure 3(d) is regular, the SIMD performance is still impaired because the memory locations for the chunk are scattered in the memory, leading to a large scatter-gather overhead. To reduce the scatter-gather overhead, we introduce the *Block Transposed Buffer* (BTB). The BTB is created by performing a block-wise *(i.e., block-by-block)* transposition of each block in the super-voxel buffer. Figure 4(a) 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 whose measurements are stored in the order of view, channel, and depth, i.e., measurements within the block are laid out along the view direction in memory. For example, measurements *a, b, c, d* in Figure 4(a) have contiguous memory locations. As shown in Figure 4(b), after transposition measurements in the chunk fall into 2 fixed-size groups, outlined by bold blue lines, and each group have contiguous memory locations. To be more specific, the number of scattered groups in a chunk equals to the 3DSV depth. Therefore, the 3DSV depth and the number of scattered groups in this example are both 2. With fewer groups and more contiguous memory accesses in each group, the scatter-gather overhead is lower and SIMD operations are more efficient.

We now show analytically the effectiveness of the BTB in increasing the SIMD performance. The average number of regular memory accesses in a chunk, denoted by N_{run}, is approximately given by

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

where N_{wh} is the number of voxels along the width and height of the 3DSV, N_b is the number of views in a block, N_d is the number of voxels along the depth of the 3DSV, and C_1 and C_2 are constants. Appendix A provides a derivation of this result along with analytical expressions for C_1 and C_2. From this equation, we observe that SIMD performance can improve by increasing block size, N_b, and block depth, N_d. Unfortunately, a larger N_d also leads to more scattered groups of the measurements and more scatter-gather overhead. Therefore, the best depth should be chosen to balance this trade-off. The experiments in Figure 7(a) corroborate this analysis by showing that a properly chosen depth delivers a 3.28 times SIMD speedup.

In addition to a properly chosen depth, the BTB design also requires a properly chosen block size. In Appendix A, we show that the fraction of essential measurements in a chunk, denoted by E_c, can be analytically approximated as

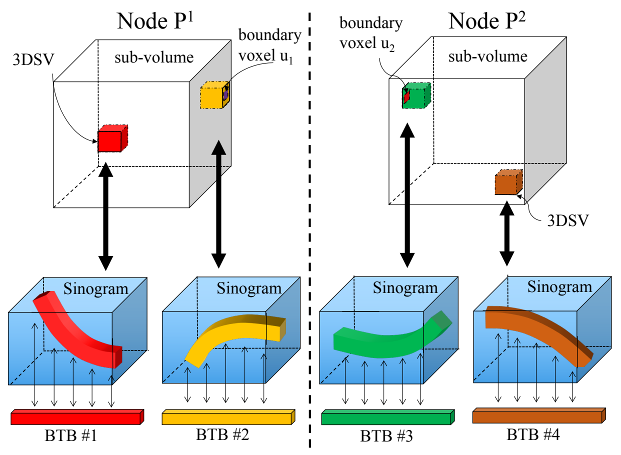
$$E_c = \frac{C_2}{N_{wh} C_1 N_b + C_2} \ . \tag{3}$$

From this equation, we observe that E_c is inversely proportional to N_b. Therefore, increasing block size not only leads to more regular memory accesses, but also additional computations. So an optimal N_b should be chosen to balance these two effects. Figures 7(b) and (c) in Section 5 give a detailed evaluation, showing that a proper block size can ensure a high SIMD speedup and a low E_c.

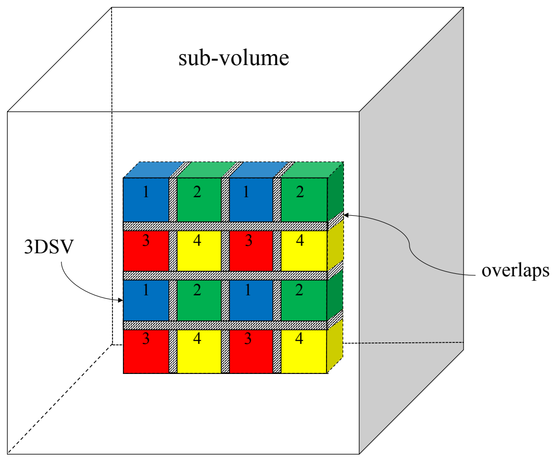
4.2 Improving Parallelism

The NU-PSV algorithm exploits three orthogonal levels of parallelism that can effectively utilize large parallel machines.

Intra-SV parallelism: data-level parallelism (SIMD vectorization) within and across multiple voxels of 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 full 3D volume.



(a)



(b)

Figure 5: (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.

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 regular memory access and a high level of SIMD vectorization.

Algorithm 2 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, $A_{*,j}$. To update a voxel-line, the intra-SV parallelism must 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 1), the values of $\theta_{1,j}$ and $\theta_{2,j}$ can be computed as summations of the partial results, denoted by $\theta^{\sim}_{1,j}$ and $\theta^{\sim}_{2,j}$, at each BTB block. Therefore in step 4, the intra-SV parallelism cycles through blocks and assigns the computations at each block to SIMD units. Then, each SIMD unit computes the partial result for its assigned voxel. 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 each SIMD unit's access is completely regularized by the organization of the BTBs.

The intra-SV parallelism, however, has a technical difficulty. Since the BLB design introduces redundant measurements, the intra-SV parallelism must ensure that the results of $\theta_{1,j}$ and $\theta_{2,j}$ are not affected by accessing redundant measurements. To address this issue, we exploit the sparse A matrix structure. More specifically, we set to zero all A matrix entries corresponding to redundant measurements, shown in yellow within chunks in Figure 4(a). For essential measurements, however, the A matrix entries are set to non-zero. Therefore, the results of $\theta_{1,j}$ and $\theta_{2,j}$, as in steps 1 and 2 of Algorithm 1, are not affected by accessing redundant measurements, since the product of zero-value A matrix elements

```
Algorithm 2 3DSV Update ($U$, $x$, $BTB_e$, $BTB_\Lambda$, $BTB_A$)
INPUT: $U$: the 3DSV to be updated
INPUT: $BTB_e$, $BTB_\Lambda$, $BTB_A$: the block-transposed buffers 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 1
6:              $\theta_{1,j} \mathrel{+}= \theta^{\sim}_{1,j}$
7:              $\theta_{2,j} \mathrel{+}= \theta^{\sim}_{2,j}$
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 1
12:         $x_j \leftarrow x_j + \alpha$
13:         $BTB_e \leftarrow BTB_e - A\alpha$
14:     end for
15: end for
```

and redundant measurements are zero. Hence, $\theta_{1,j}$ and $\theta_{2,j}$ are computed correctly.

Inter-SV parallelism. Inter-SV parallelism uses separate cores in a node to perform parallel updates of different 3DSVs within a sub-volume. Algorithm 3 summarizes how inter-SV parallelism functions. In steps 3 and 4, each 3DSV's core creates private BTBs for its 3DSV, denoted by U. Figure 5(a) shows an example of private BTBs. In step 6, each core follows Algorithm 2 and updates 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 each core's super-voxel buffer, Buf_e, must be merged and written to the full sinogram. The updated measurements, however, overlap with one other, because voxels share measurements in the sinogram (see Section 3.1). Therefore, simply copying the updated measurements back to the sinogram will lose and overwrite other cores' updates [24, 30]. To correctly write measurements to the sinogram, $Buf_{e,\Delta}$ is created in step 8 to keep track of all changes to the measurements. 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 [24, 30]. To address this, neighboring 3DSVs overlap with each other on their common boundary using halos, eliminating the partition boundary among neighboring 3DSVs. Figure 5(b) shows an example of overlapping 3DSVs, with the halo regions shaded in gray.

Although overlapping neighboring 3DSVs removes image artifacts, the simultaneous updates of voxels in the boundary regions can lead to data races. This situation occurs when the same boundary voxel is simultaneously updated by cores in step 12 of Algorithm 2. To eliminate possible races, we employ a checkerboard pattern [24] for 3DSV updates. As shown in Figure 5(b), checkerboard pattern tessellates a volume into tiles, with each tile consisting of 4 3DSVs with different colors, and with no two adjacent 3DSVs having the same color. Inter-SV parallelism cycles through the four colors in step 1 of Algorithm 3 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.

Algorithm 3 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, $Buf_e$, $Buf_\Lambda$ and $Buf_A$
4:         Block-transpose super-voxel buffers to $BTB_e$, $BTB_\Lambda$ and $BTB_A$
5:         Make a temporary copy of $Buf_e$ to $Buf_e'$
6:         3DSV Update ($U$, $S$, $BTB_e$, $BTB_\Lambda$, $BTB_A$), as in Algorithm 2
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
```

Intra-volume parallelism. Intra-volume parallelism performs parallel updates of all sub-volumes across nodes, with the i^{th} node processing the i^{th} sub-volume (S^i). Figure 5(a) shows how sub-volumes are equally distributed onto two nodes. Each node then processes its sub-volume using Algorithm 3.

Since the prior function for each voxel update in Algorithm (1) 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}.[4] An example of this dependency is shown in Figure 5(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 [23, 30], 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 one voxel, then intra-volume parallelism reduces to spatially close voxel updates, which are known to converge [1, 12].

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.

4.3 Improving Convergence

Algorithm 4 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.
1: Inter-SV Update ($S^i$) as in Algorithm 3. A max-heap is constructed.
2: repeat
3:     Re-randomize 3DSVs and divide them into $\frac{C}{\rho}$ groups.
4:     for Group $r$ from 1 to $\frac{C}{\rho}$ do
5:         Inter-SV Update ($A_g$), as in Algorithm 3.
6:         Inter-SV Update ($A_r$), as in Algorithm 3
7:         The max-heap is updated.
8:         Exchange boundary voxels asynchronously among $P^i$, $P^{i-1}$ and $P^{i+1}$
9:     end for
10: until NU-PSV converges.
```

Although the checkerboard update pattern prevents data races in inter-SV parallelism, it enforces a cyclic and fixed update order, which typically has poorer convergence speed [23].

For fast convergence, previous theoretical work proves that a combination of random updates and greedy updates has the fastest algorithmic convergence [23]. We now discuss how we apply these theoretical findings to improve the convergence of the NU-PSV algorithm. Experiments in Section 5 show that the NU-PSV algorithm provides, on average, a 1.66 times speedup due to faster convergence as compared to the same algorithm using random updates alone.

Intuitively, the NU-PSV algorithm benefits from randomized 3DSV update order because random updates ensure that computations are distributed across widely-separated groups of 3DSVs. The NU-PSV algorithm, however, also benefits from the greedy update of 3DSVs because 3DSVs in a volume require different frequencies

[4] Sub-volumes, such as S^1, that do not have adjacent boundary voxels are given imaginary boundary voxels with value zero.

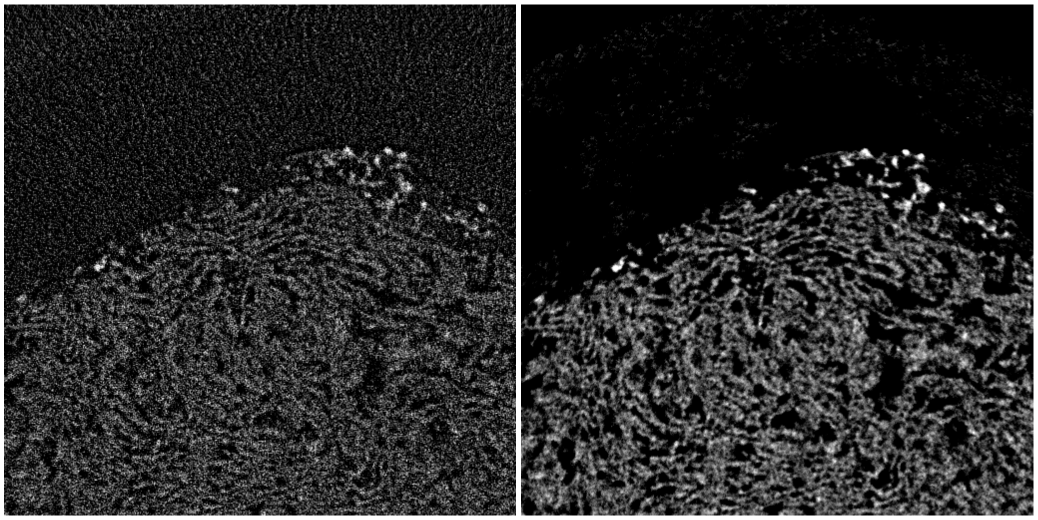


Figure 6: The left image is a slice reconstructed by the traditional reconstruction method. The right image is the same slice reconstructed by MBIR. Notice that MBIR has significantly better image quality with less noise.

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

Table 1: 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.

of updates. For example, a high radio-density 3DSV, which represents material such as bones or metals, typically requires many more updates than a low-radio density 3DSV, which represents material such as air.

Algorithm 4 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 $A_r = \frac{G\rho}{C}$ of the 3DSVs and G is the total number of 3DSVs in the sub-volume. Parameter ρ is from 0 to 1 and it controls the extent to which updates are in random update or greedy update procedure. 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 $A_g = \frac{G(1-\rho)}{C}$ of the 3DSVs on the max-heap and updates them in parallel. After the greedy update procedure, all voxels in the visited randomized group are processed and updated in parallel. The random update procedure, however, causes the max-heap to become 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 Section 4.2. In step 10, the NU-PSV algorithm repeats itself until algorithmic convergence is reached.

5 EXPERIMENTS

This section provides an experimental evaluation of how the innovations discussed in Section 4 contribute to NU-PSV's performance.

Datasets. We use three datasets to evaluate NU-PSV's performance: (1) an iron hydroxide (FeOOH) material dataset, sparsely imaged with a Zeiss Xradia 810 X-ray microscope at the United States Air Force Research Lab, (2) a slime mold biology dataset (*m vesparium*), imaged using the Advanced Light Source synchrotron at the Lawrence Berkeley National Lab, and (3) a security baggage scan dataset for a high cluttered volume, imaged with an Imatron C-300 Scanner at the Department of Homeland Security. Figure 6 shows an example slice in the iron hydroxide dataset, reconstructed by the traditional reconstruction method (Filtered Back-Projection)

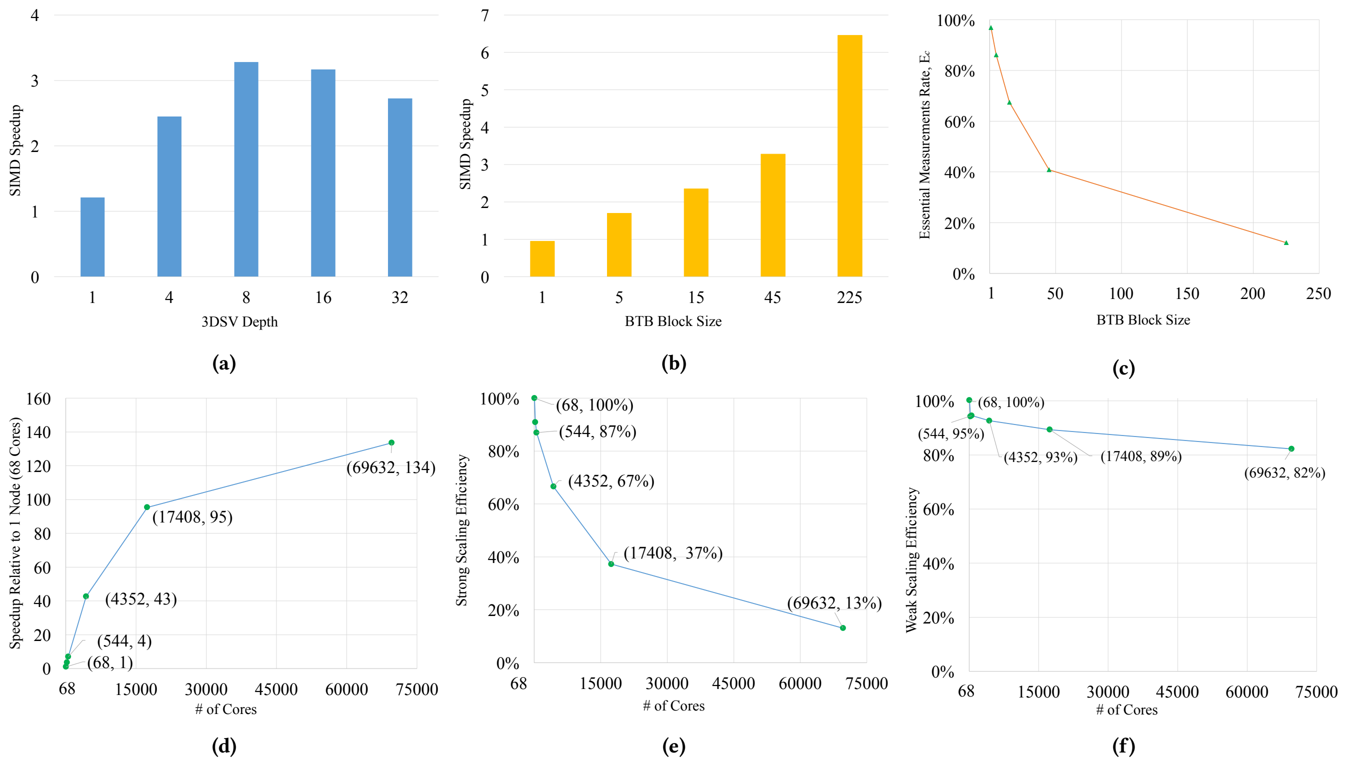


Figure 7: (a) The SIMD speedups at different 3DSV depths. (b) The SIMD speedups at different block sizes. (c) The fraction of essential measurements, E_c, at different block sizes. (d) NU-PSV's strong-scaling speedup relative to 1 node (68 cores). The pairs at each data point indicate the speedup (second numbers of the pairs) at different numbers of cores (first numbers of the pairs). (e) 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). (f) Weak scalability of NU-PSV.

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 [5] 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. Each core has a 32-KB private L1 data cache, and every two cores share a 1-MB L2 cache. All programs in this section 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 the NERSC global file system. In addition, each reported runtime is computed three times and the average is used. 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 reduces to the number of slices in the sub-volume. 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 4.3, are empirically determined to be 0.2 and 3.25 respectively for fastest convergence.

5.1 Iron Hydroxide Dataset

The iron hydroxide dataset has 1024 slices and the sinogram for each slice 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. The reconstructed volume size for this dataset is 1024^3 with a voxel resolution of $64^3 \mu m^3$.

To quantify the overall performance, NU-PSV's runtimes are compared to the performance of TIMBIR and PSV-ICD, discussed in Section 3.2. Table 1 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 NERSC's 24-hour wall time limit, noted as >86400 seconds in the table. Since TIMBIR can only scale to 8 nodes because of the algorithmic constraints

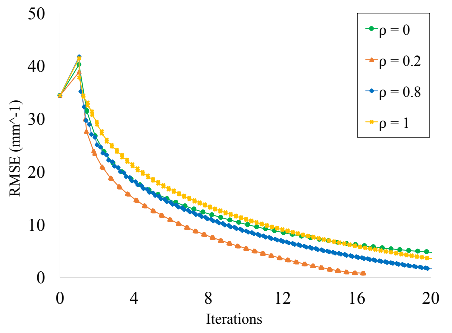


Figure 8: The algorithmic convergence rate with different ρ.

(see Section 3.2), runtimes at more than 8 nodes are reported as NA. The fourth row is the runtimes for PSV-ICD, which is a single node implementation and only scales to 68 cores. The fifth row is NU-PSV, which is 9776.0 times faster than TIMBIR and 1665.2 times faster than PSV-ICD. NU-PSV's speedups come from two different sources: the per-node speedup and the speedup due to scalability. For this dataset, the per-node speedup is greater than 83.11 relative to TIMBIR and 12.49 times relative to PSV-ICD. Moreover, NU-PSV scales to 69632 cores.

To better understand the improved per-node speedup, we start by analyzing the SIMD speedups from the 3DSV design. Figure 7(a) shows SIMD speedups at different 3DSV depths. The SIMD speedup is 1.21 with depth 1. The SIMD speedup peaks at 3.28 with depth 8, and then drops to 2.72 with depth 32. This confirms Equation (2)'s analysis that a larger 3DSV depth proportionally increases the number of regular memory accesses, often leading to better SIMD performance. At the same time, however, a too large depth leads to more scattered groups for measurements in a chunk and greater scatter-gather overhead. Therefore, the cache pressure is higher and the SIMD performance gains are negated. Measurements with the Intel Vtune Amplifier shows that the L2 cache miss rate increases from 6.4% with depth 8, to 32.3% with depth 32.

Another factor for NU-PSV's improved per-node performance is the BTB block size. Figure 7(b) shows the SIMD speedups at different block sizes. The SIMD speedup is 0.95 with a block size of 1, and 6.45 with a block size of 225. This result also confirms the theoretical analysis in Equation (2) that a larger block size increases N_{run} and leads to better SIMD vectorization.

In addition, the increased block size does not increase the scatter-gather overhead while the increased depth does increase this 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 increases to 6.45, with a lower L2 cache miss rate of 3.4%. The cache miss rate is reduced because the scatter-gather overhead is lower and spatial locality is better exploited.

Although a larger block size leads to more SIMD speedup, as indicated in Figure 7(b), a larger block size also increases redundant measurements and leads to extra computations. Figure 7(c) shows the relationship between E_c and block size, where E_c represents the fraction of essential measurements in a chunk. From the figure, we can observe that E_c is inversely proportional to block size. Therefore, the trade-off between SIMD performance gains and extra computations should be considered when choosing an optimal block size. When the block size is smaller, each core has fewer regular memory accesses, leading to a smaller SIMD performance gain. However, a smaller block size also incurs fewer extra computations. After balancing the trade-off, block size 45 is determined to be the optimal size for this dataset. Compared to the performance at block size 225, block size 45 has less SIMD speedup, as seen in Figure 7(b), but its E_c is also 3.4 times higher.

We now investigate NU-PSV's scalability. Figure 7(d) 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 (each node has 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 7(e) 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 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 7(a), NU-PSV has worse SIMD performance with 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 7(f) shows the weak scalability of NU-PSV. The data points along the plot-line are the weak scaling efficiency at different numbers of cores. We can observe that the weak-scaling efficiency overall is much higher than the strong-scaling one. At 4352 cores, the weak scaling efficiency is 93% and it drops to 82% at 69632 cores. Since the per-core work is fixed for weak-scaling experiments, SIMD performance or synchronization overhead doe not get worse at higher number of cores. The weak-scaling efficiency, however, still drops slowly when the number of cores increases because checking the algorithmic convergence in step 10 of Algorithm 4 requires a global MPI reduction across nodes.

We now investigate how the alternating random and greedy updates affect convergence. Figure 8 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 8, 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 theoretical findings [23, 33].

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

Table 2: Runtimes in seconds for the slime mold 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

Table 3: Runtimes in seconds for the security dataset.

5.2 Slime Mold Dataset

The slime mold dataset is 70 times larger than the iron hydroxide dataset and it contains 2160 slices. The sinogram for each slice 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. The reconstructed volume size is $2560 \times 2560 \times 2160$ with a voxel resolution of $1.316^3 \mu m^3$.

Table 2 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 2, TIMBIR is only scalable to 1224 cores because of the algorithmic constraint, discussed in Section 3.2, whereas NU-PSV scales to 146880 cores, with a total speedup >3483.9, compared to TIMBIR.

5.3 Security Dataset

The security dataset is 3 times smaller than the iron hydroxide dataset and it contains 440 slices. The sinogram for each slice 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. The reconstructed volume size is $512 \times 512 \times 440$, with a voxel resolution of $927.6^3 \mu m^3$.

Table 3 summarizes the reconstruction runtimes for TIMBIR, PSV-ICD and NU-PSV. TIMBIR is scalable to 272 cores because the algorithmic constraints limit the number of cores to approximately half of the number of slices. PSV-ICD is a single node implementation, scalable to 68 cores. As a comparison, NU-PSV scales to 29920 cores with a speedup of 2014.8 as compared to TIMBIR, and a speedup of 101.2 as compared to PSV-ICD.

6 IMPLICATIONS

In this section, we discuss the implications of the NU-PSV algorithm to the compressive sensing community in general.

The NU-PSV algorithm serves as a massively parallel framework for compressive sensing problems. Similar to MBIR's objective function (Equation (1)), most of the compressive sensing problems can be expressed in the form

$$\hat{x} = \underset{x \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2}(y - Ax)^T \Lambda (y - Ax) + R(x) \right\} , \qquad (4)$$

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

When the reconstructed sensor output, x, is viewed as an image, Equation (4) is the mathematical framework for image reconstruction problems[5]. But often, the sensor output, x, is a multi-dimensional array of measurements of the sample object or environment. Such applications include absolute shrinkage and selection operator (LASSO) problems in machine learning [15], geophysics sensing problems[6] and radar sensing problems [32].

To apply the NU-PSV algorithm, let 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. Then, all sub-volumes are processed in parallel across different nodes. Therefore, sub-volumes share minimal measurements in common and the inter-node communication overhead can be minimal.

Within each sub-volume, 3DSVs are processed in parallel among cores of a node 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 the 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 alternating update procedures (random update and greedy update) are used. In the random update procedure, a random group of 3DSVs is chosen and 3DSVs in the group are processed in parallel. Since 3DSVs are chosen randomly, the statistical correlation among 3DSVs are moderately low, leading to a low communication 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 be optimal.

7 CONCLUSIONS

MBIR is a widely-explored 3D image reconstruction technique that has a large and growing impact on the medical, industrial and scientific imaging communities. The slow computation speed for MBIR, however, is a bottleneck for scientific advancements in fields that use imaging, such as materials. The computational cost has also hindered its commercialization besides the medical community. This paper 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.

[5]Such as whole body CT, PET scan, MRI imaging, electron microscopy, synchrotron imaging, proton imaging and ultrasound imaging.

The technical innovations of this paper will yield immediate results to the 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 for 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.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Larry Drummy, Dr. Dilworth Parkinson and Dr. Singanallur Venkatakrishnan for providing the datasets. 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, the authors would like to thank the National Energy Research Scientific Computing Center (NERSC) for providing supercomputing resources. This research was supported by the U.S. Department of Homeland Security (DHS) under SBIR Award D15PC00110, subcontracted through High Performance Imaging LLC. Additional support was provided by the DHS ALERT Center for Excellence 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 DHS or High Performance Imaging LLC or ALERT.

REFERENCES

[1] T. M. Benson, B. D. Man, L. Fu, and J. B. Thibault. 2010. Block-Based Iterative Coordinate Descent. In *IEEE Nuclear Science Symposuim Medical Imaging Conference.* 2856–2859. DOI: http://dx.doi.org/10.1109/NSSMIC.2010.5874316

[2] T. Bicer, D. Gürsoy, V. D. Andrade, R. Kettimuthu, W. Scullin, F. D. Carlo, and I. T. Foster. 2017. Trace: A High-Throughput Tomographic Reconstruction Engine for Large-Scale Datasets. *Advanced Structural and Chemical Imaging* Vol. 3(6) (2017), 1–10. DOI: http://dx.doi.org/10.1186/s40679-017-0040-7

[3] C. A. Bouman and K. Sauer. 1996. A Unified Approach to Statistical Tomography Using Coordinate Descent Optimization. *IEEE Transactions on Image Processing* 5, 3 (March 1996), 480–492.

[4] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. 2011. Parallel Coordinate Descent for L_1-Regularized Loss Minimization. In *The 28th International Conference on Machine Learning.*

[5] The National Energy Research Scientific Computing Center. 2017. Cori Intel Xeon Phi Nodes. http://www.nersc.gov/users/computational-systems/cori/cori-intel-xeon-phi-nodes/. (2017).

[6] J. F. Claerbout and F. Muir. 1973. Robust Modeling with Erratic Data. *Geophysics* 38, 5 (1973), 826–844. DOI: http://dx.doi.org/10.1190/1.1440378

[7] N. H. Clinthorne, T. S. Pan, P. C. Chiao, W. L. Rogers, and J. A. Stamos. 1993. Preconditioning Methods for Improved Convergence Rates in Iterative Reconstructions. *IEEE Transactions on Medical Imaging* 12(1) (1993).

[8] S. Degirmenci, D. G. Politte, C. Bosch, N. Tricha, and J. A. O'Sullivan. 2015. Acceleration of Iterative Image Reconstruction for X-Ray Imaging for Security Applications. In *Proceedings of SPIE-IS&T Electronic Imaging*, Vol. 9401.

[9] DHS/ALERT. 2009. Research and Development of Reconstruction Advances in CT-based Object Detection systems. https://myfiles.neu.edu/groups/ALERT/strategic_studies/TO3_FinalReport.pdf. (2009).

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

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

[12] J. A. Fessler and D. Kim. 2011. 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.*

[13] J. W. Gibbs, K. A. Mohan, E. B. Gulsoy, A. J. Shahani, X. Xiao, C.A. Bouman, M. De Graef, and P.W. Voorhees. 2015. The Three-Dimensional Morphology of Growing Dendrites. *Scientific Reports* (2015).

[14] S. Ha and K. Mueller. 2015. An Algorithm to Compute Independent Sets of Voxels for Parallelization of ICD-based Statistical Iterative Reconstruction. In *The 13th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine.*

[15] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. 2008. A Dual Coordinate Descent Method for Large-Scale Linear SVM. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08).* ACM, New York, NY, USA, 408–415. DOI: http://dx.doi.org/10.1145/1390156.1390208

[16] J. Hsieh, , B. Nett, Z. Yu, K. Sauer, J.-B. Thibault, and C. A. Bouman. 2013. Recent Advances in CT Image Reconstruction. *Current Radiology Reports* 1, 1 (2013), 39–51. DOI: http://dx.doi.org/10.1007/s40134-012-0003-7

[17] P. Jin, C. A. Bouman, and K. D. Sauer. 2013. A Method for Simultaneous Image Reconstruction and Beam Hardening Correction. In *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC).* 1–5.

[18] P. Jin, E. Haneda, C. A. Bouman, and K. D. Sauer. 2012. 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.*

[19] S. J. Kisner, E. Haneda, C. A. Bouman, S. Skatter, M. Kourinny, and S. Bedford. 2012. Model-Based CT Reconstruction from Sparse Views. In *Second International Conference on Image Formation in X-Ray Computed Tomography.* 444–447.

[20] B. D. Man, S. Basu, J-B. Thibault, J. Hsieh, J. A. Fessler, C. A. Bouman, and K. Sauer. 2005. A Study of Different Minimization Approaches for Iterative Reconstruction in X-ray CT. In *IEEE Nuclear Science Symposium*, Vol. 5. 2708–2710.

[21] K. A. Mohan, S. V. Venkatakrishnan, J. W. Gibbs, E. B. Gulsoy, X. Xiao, M. D. Graef, P. W. Voorhees, and C. A. Bouman. 2015. TIMBER: A Method for Time-Space Reconstruction from Interlaced Views. *IEEE Transactions on Computational Imaging* 1, 2 (June 2015), 96–111.

[22] Y. Nesterov. 2012. Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems. *SIAM Journal on Optimization* 22, 2 (2012), 341–362. DOI: http://dx.doi.org/10.1137/100802001

[23] J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke. 2015. Coordinate Descent Converges Faster with the Gauss-southwell Rule Than Random Selection. In *The 32nd International Conference on Machine Learning (ICML'15).* 1632–1641. http://dl.acm.org/citation.cfm?id=3045118.3045292

[24] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff. 2017. Model-Based Iterative CT Image Reconstruction on GPUs. In *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '17).* ACM, New York, NY, USA, 207–220. DOI: http://dx.doi.org/10.1145/3018743.3018765

[25] K. Sauer, S. Borman, and C. Bouman. 1995. Parallel Computation of Sequential Pixel Updates in Statistical Tomographic Reconstruction. *IEEE Intl. Conf. on Image Processing* 3 (October 23-25 1995).

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

[27] J. B. Thibault, K. D. Sauer, C. A. Bouman, and J. Hsieh. 2007. A Three-Dimensional Statistical Approach to Improved Image Quality for Multi-Slice Helical CT. *Medical Physics* 34(11) (2007).

[28] X. Wang. 2017. *High Performance Tomography.* Ph.D. Dissertation. School of Electrical and Computer Engineering, Purdue University, the United States.

[29] X. Wang, K. A. Mohan, S. J. Kisner, C. A. Bouman, and S. P. Midkiff. 2016. Fast Voxel Line Update for Time-Space Image Reconstruction. In *The 41st IEEE International Conference on Acoustics, Speech and Signal Processing.*

[30] X. Wang, A. Sabne, S. J. Kisner, A. Raghunathan, C. A. Bouman, and S. P. Midkiff. 2016. High Performance Model Based Image Reconstruction. In *21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.*

[31] S. J. Wright. 2015. Coordinate Descent Algorithms. *Math. Program.* 151, 1 (June 2015), 3–34. DOI: http://dx.doi.org/10.1007/s10107-015-0892-3

[32] L. Wu, P. Babu, and D. P. Palomar. 2017. Cognitive Radar-Based Sequence Design via SINR Maximization. *IEEE Transactions on Signal Processing* 65(3) (2017). DOI: http://dx.doi.org/10.1109/TSP.2016.2621723

[33] Z. Yu, J-B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh. 2011. Fast Model-Based X-Ray CT Reconstruction Using Spatially Nonhomogeneous ICD Optimization. *IEEE Transactions on Image Processing* 20(1) (2011).

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

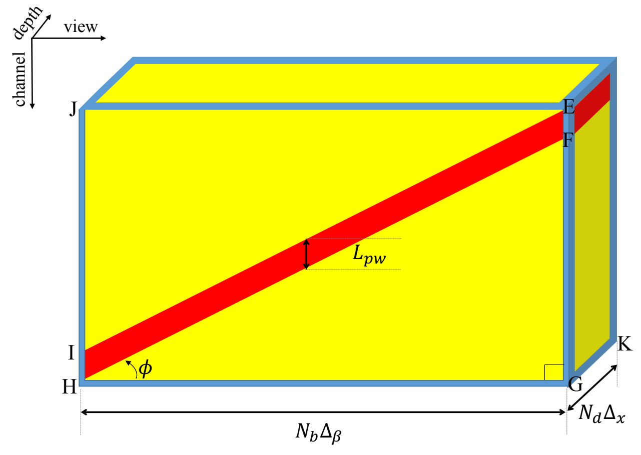


Figure 9: Demonstrates the dimensions of a chunk. The voxel-line trace is colored in red and the redundant measurements are colored in yellow.

A APPENDIX: EQUATIONS' DERIVATIONS

Figure 9 shows a chunk in a BTB, where the voxel-line trace (essential measurements) is colored in red and redundant measurements are colored in yellow. The chunk length, HG, equals to $N_b\Delta_\beta$, 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, GK, is $N_d\Delta_x$, where N_d is the depth of the 3DSV and Δ_x is the voxel width; the chunk height, EG, is $L_{pw} + m(\phi)N_b\Delta_\beta$, where L_{pw} is the average voxel trace width, ϕ is the angle between the voxel trace and the view direction, $m(\phi)$ is the average voxel trace absolute slope. Relating to Figure 9, 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:

$$V_c = N_b\Delta_\beta N_d\Delta_x\left(L_{pw} + m(\phi)N_b\Delta_\beta\right) \tag{5}$$

Then, the average number of regular memory accesses in a chunk, denoted as N_{run}, can then be computed as:

$$N_{run} = \frac{V_c}{\Delta_\beta\Delta_x\Delta_d} \tag{6}$$

where Δ_d is the detector channel spacing, $\Delta_\beta\Delta_x\Delta_d$ is the size of a single entry in the BTB, and $\frac{V_c}{\Delta_\beta\Delta_x\Delta_d}$ represents the number of element entries in a chunk. By plugging the expression of V_c to Equation (6), we can then get:

$$N_{run} = \frac{N_dN_b\left(L_{pw} + m(\phi)N_b\Delta_\beta\right)}{\Delta_d}. \tag{7}$$

In this equation, N_d and N_b are known constants, chosen to optimize computing performance. Δ_β and Δ_d are also known constants about the dataset. L_{pw} and $m(\phi)$, however, are unknown parameters. To compute Equation (7), we must first compute these two parameters.

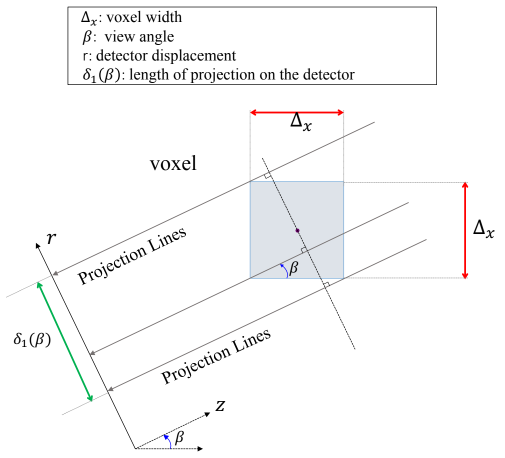


Figure 10: Shows a voxel modeled as a square. The length of projection, $\delta_1(\beta)$, is shown as the green bar.

Computing parameter L_{pw} To compute L_{pw}, we model a voxel to be a square with voxel width Δ_x, as shown in Figure 10. In addition, we denote $\delta_1(\beta)$, shown as a green bar in Figure 10, as the length of projection on the X-ray detector at view angle β. $\delta_1(\beta)$ can be computed as in [28]:

$$\delta_1(\beta) = \begin{cases}\sqrt{2}\Delta_x\cos(\frac{\pi}{4} - \beta), & \text{if } \beta \in [0, \frac{\pi}{2})\\ \sqrt{2}\Delta_x\sin(\beta - \frac{\pi}{4}), & \text{if } \beta \in [\frac{\pi}{2}, \pi)\end{cases} \tag{8}$$

Unfortunately, Equation (8) is not ideal because $\delta_1(\beta)$ must be approximated to be a multiple of Δ_d in real applications.

To offset this error, a constant term Δ_d is added to Equation (8) and the new equation becomes:

$$\delta_1(\beta) \approx \begin{cases}\sqrt{2}\Delta_x\cos(\frac{\pi}{4} - \beta) + \Delta_d, & \text{if } \beta \in [0, \frac{\pi}{2})\\ \sqrt{2}\Delta_x\sin(\beta - \frac{\pi}{4}) + \Delta_d, & \text{if } \beta \in [\frac{\pi}{2}, \pi)\end{cases} \tag{9}$$

With Equation (9), L_{pw} can then be computed to be the average value for $\delta_1(\beta)$, shown as:

$$L_{pw} = \frac{\int_0^\pi \delta_1(\beta)\,d\beta}{\pi} \approx \frac{4\Delta_x}{\pi} + \Delta_d \tag{10}$$

Computing parameter $m(\phi)$ For the j^{th} voxel in a slice, illustrated as a red square in Figure 11(a), we denote its coordinate as (x_j, y_j). In addition, we denote its voxel trace amplitude in the sinogram at view angle β as $r_j(\beta)$, shown as a yellow bar in Figure 11(b). Analytically, $r_j(\beta)$ can be expressed as [28]:

$$r_j(\beta) = y_j\cos\beta - x_j\sin\beta \tag{11}$$

and the voxel trace slope in the sinogram at view angle β is then:

$$r_j^{'}(\beta) = -y_j\sin\beta - x_j\cos\beta \tag{12}$$

Therefore, the average absolute slope for a voxel trace in the sinogram space, denoted as $\tilde{m}$, can be computed as:

$$\tilde{m} = \frac{\int_{\frac{-N_x}{2}}^{\frac{N_x}{2}}\int_{\frac{-N_x}{2}}^{\frac{N_x}{2}}\int_0^\pi |r_j^{'}(\beta)|\,d\beta\,dx_j\,dy_j}{N_xN_x\pi} \tag{13}$$

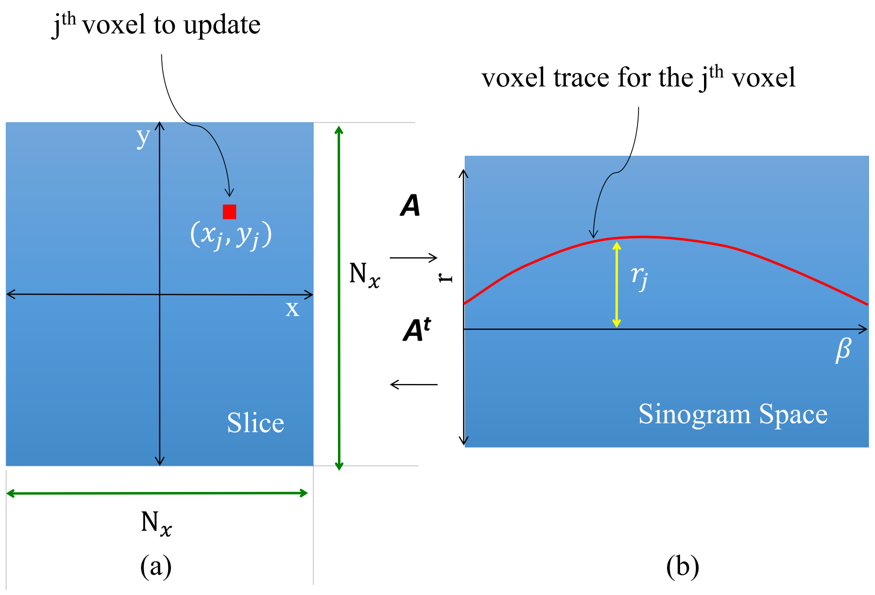


Figure 11: (a) The red square in the slice is the j^{th} voxel, whose coordinate is (x_j, y_j). (b) The red trace is the measurements for the j^{th} voxel. The yellow bar r_j represents the voxel trace amplitude in the sinogram space.

Where N_x is the slice dimension. To simplify Equation (13), we use polar coordinate and we let $x_j = -\gamma \cos\beta$, $y_j = -\gamma \sin\beta$ and $\gamma = \sqrt{x_j^2 + y_j^2}$. Therefore,

$$\tilde{m} = \frac{8\int_0^{\frac{\pi}{4}} \int_0^{\frac{N_x}{2\cos\beta}} 2\gamma^2 \, d\gamma \, d\beta}{N_x N_x \pi} = \frac{N_x}{3\pi}\left(\sqrt{2} + \ln(1+\sqrt{2})\right) \tag{14}$$

Similar as before, Equation (14) must compensate for errors. Therefore, a constant term is added to Equation (14) and $\tilde{m}$ is approximated to be:

$$\tilde{m} \approx \frac{N_x}{3\pi}\left(1 + \sqrt{2} + \ln(1+\sqrt{2})\right) \tag{15}$$

After measurements are copied from the sinogram space to a BTB, all voxel traces are flattened with a much smaller amplitude and slope (see Section 4.1 for more detail). To calculate the voxel trace average absolute slope in the BTB, $m(\phi)$, a 3DSV can be viewed as a slice, whose length and height is $N_{wh}\Delta_x$, where N_{wh} is the number of voxels along the width and height of the 3DSV. Therefore, $m(\phi)$ can be calculated by replacing N_x with $N_{wh}\Delta_x$ in Equation (15):

$$m(\phi) \approx \frac{N_{wh}\Delta_x}{3\pi}\left(1 + \sqrt{2} + \ln(1+\sqrt{2})\right) \tag{16}$$

After plugging Equations (10) and (16) into Equation (7), we can then get:

$$N_{run} = \left(\frac{N_{wh}\Delta_x N_b \Delta_\beta}{3\pi\Delta_d}\left(1 + \sqrt{2} + \ln(1+\sqrt{2})\right) + \frac{4\Delta_x}{\pi\Delta_d} + 1\right) N_d N_b \tag{17}$$

Since Δ_β can also be computed as $\frac{\pi}{N_v}$ [28], the full analytical expression for N_{run} becomes:

$$N_{run} = \left(\frac{N_{wh}\Delta_x N_b}{3\Delta_d N_v}\left(1 + \sqrt{2} + \ln(1+\sqrt{2})\right) + \frac{4\Delta_x}{\pi\Delta_d} + 1\right) N_d N_b \tag{18}$$

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

$$N_{run} = (N_{wh}C_1 N_b^2 + C_2 N_b)N_d \tag{19}$$

where $N_{wh}C_1N_b^2N_d$ is the number of regular memory access for redundant measurements and $C_2N_bN_d$ is the number of regular memory access for essential measurements. The percentage of essential measurements, E_c, can then be computed as the ratio of essential measurement entries to N_{run}:

$$E_c = \frac{C_2 N_b N_d}{N_{run}} = \frac{C_2}{N_{wh}C_1 N_b + C_2} \tag{20}$$