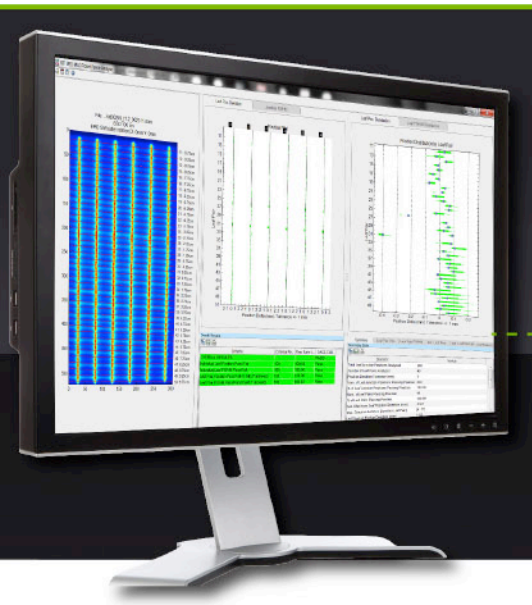
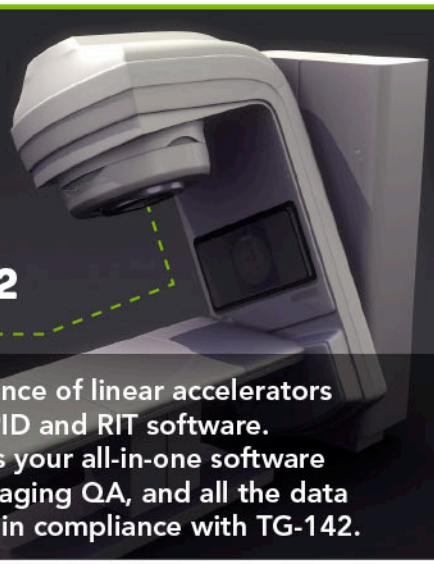
RITG142 PERFORM COMPREHENSIVE TG-142 QA WITH RIT SOFTWARE

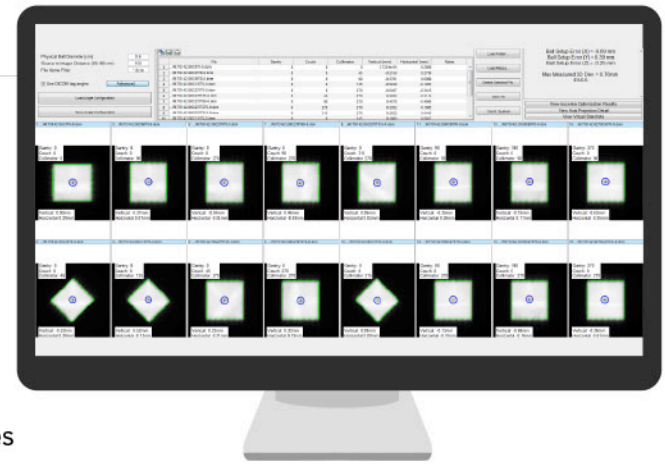


THE SINGLE-VENDOR SOFTWARE SOLUTION FOR EVERY QA TEST RECOMMENDED IN TG-142



Perform comprehensive quality assurance of linear accelerators with confidence and ease, using an EPID and RIT software. Use either RITG142 or RIT *Complete* as your all-in-one software product for Machine QA, MLC QA, Imaging QA, and all the data tracking and trending you need to be in compliance with TG-142.

Daily, Monthly & Annual LINAC Machine QA Tests



- Enhanced 3D Winston-Lutz (Isocenter Optimization)
- RIT's New Fully-Automated Star Shot Analysis
- Stereotactic Alignment (2D Winston-Lutz) Test
- Radiation/Light Field Coincidence
- Stereotactic Cone Profiles
- Field Alignment Test
- Electron Energy (TG-25)
- Asymmetric Field/Matchline
- Quick Flatness and Symmetry
- Water Tank Beam Measurement Analysis
- Depth Dose Profiles, Cross Profiles & Orthogonal Profiles

Fast and Easy Quantitative MLC QA

- EPID Picket Fence Test
- Leaf Speed Tests for Varian and Elekta
- Hancock Tests for Elekta Machines
- Automated Varian RapidArc® Tests (Tests 0.1, 0.2, 0.2 HD, 1.1, 1.1 HD, 1.2, 1.2 HD, 2 & 3)
- Bayouth MLC Test
- TG-50 Picket Fence Test
- MSK Leaf Test
- Varian DMLC Test Patterns
- MLC Transmission Analysis

One-Click Instant Imaging QA

- **Planar MV (EPID) Imager:** EPID phantom, Las Vegas, PTW EPID QC, and Standard Imaging QC-3 Phantoms
- **Planar kV Imaging:** IBA Primus® L, PTW NORMI®-4, Leeds TOR-18 FG, and Standard Imaging QC-kV1 Phantoms
- **CBCT/MVCT:** Catphan® 504 and 604 for Varian, Catphan® 503 Elekta XVI & Siemens MVCT Phantoms
- **Daily IGRT QA:** ISOCube™ Phantom: kV-MV Isocenter Coincidence, CBCT Isocenter Coincidence, kV Collimation, MV Collimation/ Light Field & 6 Degree-of-Freedom Couch Tests

RIT software does more than perform Machine QA, MLC QA, and Imaging QA measurements. The software also simplifies the process of creating a routine QA program at your facility, streamlines the reporting process for accreditation, and provides you with a better understanding of measurement performance across multiple machines and facilities.

CLICK HERE TO REQUEST A PERSONAL DEMO TODAY!

RIT®

RADIMAGE.COM

+1.719.351.1399 // sales@radimage.com // Connect with us on social media @RIT4QA

Accelerating iterative coordinate descent using a stored system matrix

Scott S. Hsieh[a] and John M. Hoffman
Department of Radiological Sciences, UCLA, Los Angeles, CA 90024, USA

Frederic Noo
Department of Radiology and Imaging Sciences, University of Utah, Salt Lake City, UT 84108, USA

(Received 13 November 2018; revised 11 March 2019; accepted for publication 5 April 2019; published 6 December 2019)

Purpose: The computational burden associated with model-based iterative reconstruction (MBIR) is still a practical limitation. Iterative coordinate descent (ICD) is an optimization approach for MBIR that has sometimes been thought to be incompatible with modern computing architectures, especially graphics processing units (GPUs). The purpose of this work is to accelerate the previously released open-source FreeCT_ICD to include GPU acceleration and to demonstrate computational performance with ICD that is comparable with simultaneous update approaches.
Methods: FreeCT_ICD uses a stored system matrix (SSM), which precalculates the forward projector in the form of a sparse matrix and then reconstructs on a rotating coordinate grid to exploit helical symmetry. In our GPU ICD implementation, we shuffle the sinogram memory ordering such that data access in the sinogram coalesce into fewer transactions. We also update N_S voxels in the xy-plane simultaneously to improve occupancy. Conventional ICD updates voxels sequentially ($N_S = 1$). Using $N_S > 1$ eliminates existing convergence guarantees. Convergence behavior in a clinical dataset was therefore studied empirically.
Results: On a pediatric dataset with sinogram size of $736 \times 16 \times 13860$ reconstructed to a matrix size of $512 \times 512 \times 128$, our code requires about 20 s per iteration on a single GPU compared to 2300 s per iteration for a 6-core CPU using FreeCT_ICD. After 400 iterations, the proposed and reference codes converge within 2 HU RMS difference (RMSD). Using a wFBP initialization, convergence within 10 HU RMSD is achieved within 4 min. Convergence is similar with N_S values between 1 and 256, and $N_S = 16$ was sufficient to achieve maximum performance. Divergence was not observed until $N_S > 1024$.
Conclusions: With appropriate modifications, ICD may be able to achieve computational performance competitive with simultaneous update algorithms currently used for MBIR. © *2019 American Association of Physicists in Medicine* [https://doi.org/10.1002/mp.13543]

Key words: iterative reconstruction, GPU acceleration, iterative coordinate descent

1. INTRODUCTION

Statistical iterative reconstruction (SIR) has emerged as a staple of modern CT.[1] While SIR has a long history, including similarity with the algebriac reconstruction technique (ART) or simultaneous algebraic reconstruction technique (SART) algorithms used in some of the first CT work,[2] it was not until recently[3] that they were routinely included in commercial scanners.

Statistical iterative reconstruction has several potential advantages and has been implemented specifically for the purpose of dose reduction. The dose reduction possible from SIR varies with the study.[4–6] Unfortunately, automated methods for assessing dose reduction are difficult and traditional MTF and NPS metrics are not useful.[7] Clinical studies comparing SIR to filtered backprojection (FBP) have shown an advantage for SIR at very low doses.[8,9] Dose reduction may be possible because SIR is able to use underlying data statistics.[10] While care is needed to avoid missed findings, a modest reduction in protocol mAs can be used in conjunction with SIR to reduce radiation dose.

A wide variety of SIR approaches are available. One form of SIR poses reconstruction as an optimization problem over hundreds of millions of variables,[1] which is very time consuming to solve. We will refer to this form of SIR, which relies on optimization of an objective function, as model-based iterative reconstruction (MBIR) to distinguish it from other SIR approaches which are currently being used. To avoid the expensive optimization of MBIR, many commercial SIR algorithms are proprietary hybrids that strike a compromise between model accuracy, convergence, and computational speed. For example, the original FBP algorithm[2] can be adapted to include selective denoising.[11] While one might expect a dichotomy between analytic and iterative reconstruction, the current reality is that commercial reconstruction algorithms lie on a spectrum between analytic FBP and MBIR. Fast alternatives to MBIR have appeared, some of which rely on drastically different principles such as the encoding of normal tissue patterns through an image patch dictionary,[12] using other patches in the reconstructed volume as a prior,[13] or using neural networks.[14] We will focus this work on optimization-based MBIR, where computation is still a significant obstacle to adoption.

Broadly speaking, implementations of MBIR can be divided into two categories: "sequential update", modifying a

single voxel at a time (also known as iterative coordinate descent, ICD), or "simultaneous update", where the entire volume is updated simultaneously. Over recent years, parallel computing platforms, especially graphics processing units (GPUs), have become ubiquitous. These platforms seem to favor the simultaneous update strategy because they afford easy and massive parallelization. Simultaneous update of SIR typically includes a gradient descent core that is further augmented with techniques such as ordered subsets,[15] momentum,[16] or use of the dual space.[17] In contrast, accelerations of ICD have been studied in relatively little detail. Demonstrations of ICD on the GPU have been performed only recently. Sabne et al. showed a fast implementation of ICD on two-dimensional datasets using a wide variety of GPU optimizations.[18] Li et al. extended this work to a three-dimensional axial scan.[19] Ha and Mueller demonstrated ICD on a cone beam CT dataset acquired in an axial fashion.[20] To our knowledge, efficient reconstruction using GPU-based ICD has not been demonstrated for diagnostic helical CT.

Our group has recently released FreeCT_ICD, an open-source reconstruction package that implements ICD on diagnostic CT data. Together with efforts to release a repository of open-source sinogram data,[21] the FreeCT packages[22,23] could enable reproducible, extensible community development of reconstruction software. However, FreeCT_ICD currently requires several hours to converge to a solution, hampering further development efforts as well as the long-term viability of using FreeCT_ICD as a tool for CT imaging research. The purpose of this work is to report on interim results of the optimization of GPU ICD, an adaptation of FreeCT_ICD that uses GPU hardware. Another goal is to demonstrate that GPU-based ICD is possible and practical for third-generation helical CT. In contrast to conventional wisdom, accelerated ICD could be competitive with simultaneous update techniques.

In Section 2, we will review the algorithm of FreeCT_ICD and describe its translation into GPU codes and major design decisions. In Section 3, we will compare the proposed GPU code with the reference FreeCT_ICD on a pediatric dataset. We conclude in Section 4 with a discussion and future outlook.

2. METHODS

2.A. FreeCT_ICD and the stored system matrix

Our code is modeled after our reference software package, FreeCT_ICD. One of the design goals for our accelerated GPU ICD code is to reach numerical agreement with the reference software package in the limit of many iterations. We briefly review the design of FreeCT_ICD here. The reader is referred to Ref. [22] for more details.

Like other optimization-based MBIR packages, FreeCT_ICD implements a solution to the following minimization problem:

$$\hat{x} = \arg\min_{x}\{(y - Ax)^T D(y - Ax) + R(X)\} \quad (1)$$

where y is the measured sinogram, x is the reconstructed image, R is the regularizer, and D is a matrix of data weights. The current version of FreeCT_ICD does not implement data weights, and D is therefore the identity matrix. The ICD approach to solving Eq. (1) has been described elsewhere, and for the sake of brevity, we will not repeat it here. Thibault et al. applied it to helical CT with a three-dimensional regularizer[1] and described the process in detail. Its adaptation in the FreeCT package has previously been described.[22]

In contrast to a conventional approach that reconstructs onto the Cartesian grid, FreeCT_ICD performs the reconstruction in a frame of reference that rotates alongside the helical source trajectory.[24] This rotating frame of reference has been proposed in the past for computational acceleration[25,26] and it creates symmetry in the system matrix A, allowing it to be precomputed[24,27] as a stored system matrix (SSM). The resulting SSM requires memory storage in the order of 10 GB,[27] but in some protocols using flying focal spot or low pitch the SSM may exceed 50 GB. We assume in this work that the SSM is small enough such that it can reside within system RAM, but we do not assume that the SSM can fit in GPU RAM. At the time of this writing, appropriately configured workstations can have an order of magnitude greater system RAM than GPU RAM.

An advantage of the SSM is that more complex projectors could be used, without limitation to geometries that can be computed on-the-fly.[28,29] A disadvantage of the rotating-grid SSM approach employed in this work is the need for eventual derotation to a Cartesian frame of reference before storage in picture archiving and communication systems (PACS), leading to possible resolution loss.

2.B. Parallel update strategy

Equation (1) is a strictly convex and differentiable function for common choices of the regularizer function and converges if each voxel (or coordinate) is separately and sequentially optimized as a one-dimensional function.[30] Updating every voxel in the desired reconstruction grid once is considered an iteration of ICD. On the other hand, sequential updates do not provide enough parallelism for the GPU to work efficiently. To increase computational performance, multiple voxels must be updated simultaneously. We update multiple voxels both in the z-direction and in the xy-plane.

Updates of multiple voxels in the z-direction have been explored by other groups.[20,31] These parallel updates are safe if the voxels in question are not interacting: that is, if no ray intersects both voxels and the regularizer does not connect the two voxels, a modification of one voxel's value in Eq. (1) does not affect the optimization of the second voxel. In this case, updating both voxels simultaneously is mathematically equivalent to sequential update. We updated every other voxel in the z-direction, observing small (but nonzero) interactions between these voxels. The amount of interaction may depend on the acquisition protocol, including the helical pitch and slice thickness, but we did not investigate these variations in this aspect. Instead, we used only the parameters in Table I, including the common pitch value of 1.

Updating multiple voxels in the xy-plane simultaneously is much more likely to result in non-trivial interactions and can

TABLE I. Parameters of the pediatric dataset and reconstruction. "Average entries per voxel in SSM" refers to the average number of nonzero entries of the stored system matrix (SSM), which is the number of rays that intersect the average voxel according to the model of forward projector physics.

CT scanner	Definition AS
Detector size (pixels)	736 × 16
Views per rotation	1152
Total number of views	13860
Helical pitch	1.0
Reconstruction matrix	512 × 512 × 128
Voxel dimensions (mm)	0.98 × 0.98 × 1.5
Average entries per voxel in SSM	6952
Maximum entries per voxel in SSM	10252
Quadratic regularizer strength λ	0.1
Total stored system matrix size	21 GB
Reconstruction GPU	GTX 1080

lead to divergence. Nonetheless, this strategy has been deployed effectively on an empirical basis in other recent work.[18] We chose to update N_S voxels in the xy-plane simultaneously, where N_S is a tunable parameter. Care must be taken in choosing N_S because small values of N_S will create insufficient parallelism, and large values of N_S can cause divergence. One of the goals of this work was to understand if values of N_S exist such that parallelism is saturated but divergence does not occur.

Within the xy-plane, various strategies have been proposed for selecting the voxel update ordering, including lexicographic ordering (left-to-right, top-to-bottom),[22] random updates,[1] and a probabilistic update criteria that favors voxels likely to change.[32] In this work, we use a random update strategy where we create a traversal order that visits each voxel once but in random order. This could be created, for example, by starting with a lexicographical ordering and then applying a random permutation. A single iteration consists of visiting each voxel once. The traversal order is random but is not changed between iterations. We briefly investigated using a lexicographical update ordering but found that this could support only small values of N_S, because adjacent voxels would often be updated simultaneously, ignoring their strong interaction terms.

For a N_Z-slice reconstruction, a set of $N_S\left(\frac{N_Z}{2}\right)$ voxels are optimized simultaneously. For each of these voxels, the program optimizes Eq. (1) for each of the voxels in a one-dimensional sense assuming no other voxels change. The $\frac{N_Z N_S}{2}$ voxels are then updated to their new optimal values simultaneously before proceeding to the next set of voxels.

2.C. Code architecture

Our code is structured into three kernel launches: backprojection, optimization, and forward projection. A CUDA kernel is a set of instructions that is initiated from the CPU but which executes on the GPU. Synchronization is guaranteed at the end of each kernel launch assuming a single stream is used because all cores in the GPU finish processing and flush memory updates. Each series of the three kernel launches is performed for each set of $\frac{N_Z N_S}{2}$ voxels. When all voxels in the volume have been updated once, the iteration is considered complete.

CUDA kernels are called using blocks of threads, also called threadblocks. A threadblock is assigned to a streaming multiprocessor (SM). Modern GPUs house approximately 30 physical SMs. Depending on the number of threads present, full utilization of a SM may require multiple assigned threadblocks. We structured our code such that each threadblock operates on a single (x, y) column, with member threads indexing over different z-coordinates. In backprojection and forward projection, each threadblock iterates over only a fraction of the available views. This allows several threadblocks (about 10) to be assigned to each (x, y) column. With this architecture, expected $N_S \sim 16$ to yield approximately 160 threadblocks, which would be sufficient to saturate the SMs on the GPU.

The first kernel to be called is the backprojector. In the context of ICD, this means that this kernel calculates the data fidelity term, $(y - Ax)^T D(y - Ax)$ of Eq. (1), for a specific voxel. The expression $(y - Ax)^T D(y - Ax)$ can be recognized as a quadratic form in x, and when regarded as a function of a single voxel, it reduces to a simple parabola. Therefore, the backprojector calculates the linear and quadratic terms of the parabola. The constant term of the parabola is ignored because it is irrelevant for the purpose of optimization. While we use the term "backprojection" in analogy with its usage elsewhere in CT physics, a standard backprojector of the residual sinogram computes only a gradient term. In this work, the backprojector additionally computes the quadratic term, which is necessary to characterize the cost function and perform a full minimization. It should be pointed out that both the backprojection and forward projection are encoded using the SSM, which is the A matrix. To accelerate access from the SSM, the backprojector first caches entries from the SSM in shared memory. Individual threads accumulate the backprojection over several views and then write to global memory using atomic operations, which ensures that only one thread can update a value at a time.

The optimization kernel calculates the optimal value of the voxel using both the regularizer and the data fidelity parabola. Because we used a quadratic regularizer, the sum of the regularizer and data fidelity term is simply the sum of the individual parabolas, and the minimizer of the parabola can be calculated analytically. If a nonquadratic regularizer were used, brute force search or the bisection algorithm could be used to find the optimum. The bisection algorithm would exploit the fact that a convex objective function has only one minimum to progressive narrowly the search region. When generalized to a GPU with many parallel processors, a generalized k-section algorithm may be able to more efficiently find the optimum. We made no attempt to accelerate this kernel because the execution times of backprojection and forward projection are an order of magnitude slower.

The forward projector kernel updates the sinogram entries using the new optimal value of the kernel. Because multiple threads could conceivably write to the same sinogram entry,

atomics are used. From a memory bandwidth perspective, the forward projector is the slowest kernel because each sinogram entry must be first read and then written to global memory. The required memory bandwidth is twice that of the backprojector, and the use of atomics imposes a further speed penalty.

In the first iteration of ICD, the error sinogram $y - Ax$ is calculated as the difference between the raw data sinogram and the forward projection of the starting guess. The backprojector and optimization kernels are skipped. The original raw data sinogram, y, is discarded in subsequent iterations, as only the error sinogram $y - Ax$ is used in Eq. (1).

As previously described, the SSM is typically too large to fit into the GPU. Chunks of the SSM are sequentially transferred onto the GPU. Each chunk represents a subset of the voxels in the xy-plane, and when the entire subset of voxels is updated, the next chunk is transferred onto the GPU. This assumes the entire SSM can fit into CPU RAM. If the SSM is too large and must be cached to disk, the read times for the SSM will create a significant bottleneck. Other investigators have modified the SSM to allow storage on the GPU,[27] but this is equivalent to modifying the forward projector physics and may not be desirable. The transfer times of the SSM chunks onto the GPU can be masked using CUDA streams, but this was not implemented in our current code because the acceleration potential would be modest. For an SSM of 10 GB, masking transfer times would improve performance by about 1 s per iteration.

2.D. Memory access strategy

The single most important optimization in our code is to coalesce memory accesses. Data are retrieved from GPU RAM in transactions of 128 adjacent bytes. If only a single 4-byte variable is used, then the remaining 124 bytes are discarded and the memory bandwidth of the GPU is underutilized. In most codes, the sinogram is stored with the channel direction, row direction, or view direction as the fastest changing index. Our threadblock architecture is such that, in most cases, when one thread requests access to a ray through a voxel at (x, y, z), the next thread will request the corresponding ray through the voxel at (x, y, z + 2) considering that we are reconstructing every other voxel in the z-direction. By helical symmetry, the two rays will have identical channel and row indices, but their view indices will be different. We recall that in the derivation of the SSM using helical symmetry, assuming a reconstructed slice thickness of dz and a table translation of dv per view, the separation in the view direction between adjacent slices is the integer $N_{sep} = dz/dv$. Therefore, the crucial optimization in our code is to shuffle the sinogram data such that the fastest changing index is $2N_{sep}$ in the view direction. A similar idea has been used elsewhere.[25,26] This ensures that sequential threads access sequential data in memory and drastically increases the fraction of memory access that is usable to the program. Note that this strategy cannot be used for axial scans. More generally, the strategy of precomputing and storing the system matrix is impractical for most axial reconstructions.

Figure 1 illustrates this memory shuffle technique graphically. The impact of this technique is significant in codes that are limited by memory bandwidth. In the most extreme case, a program entirely bound by memory bandwidth could be accelerated by a factor of 32, because 128 bytes (32 floats) will be retrieved but only one is used. Figure 1 depicts a geometry where the acceleration factor would be up to 16, because each transaction would result in two useful floats. In practice, considering cache effects, misalignment, and other bottlenecks on computation, the acceleration factor will be smaller but may still approach an order of magnitude.

2.E. Experiments

We used a pediatric dataset in our evaluation experiments. The details of this dataset and the relevant reconstruction

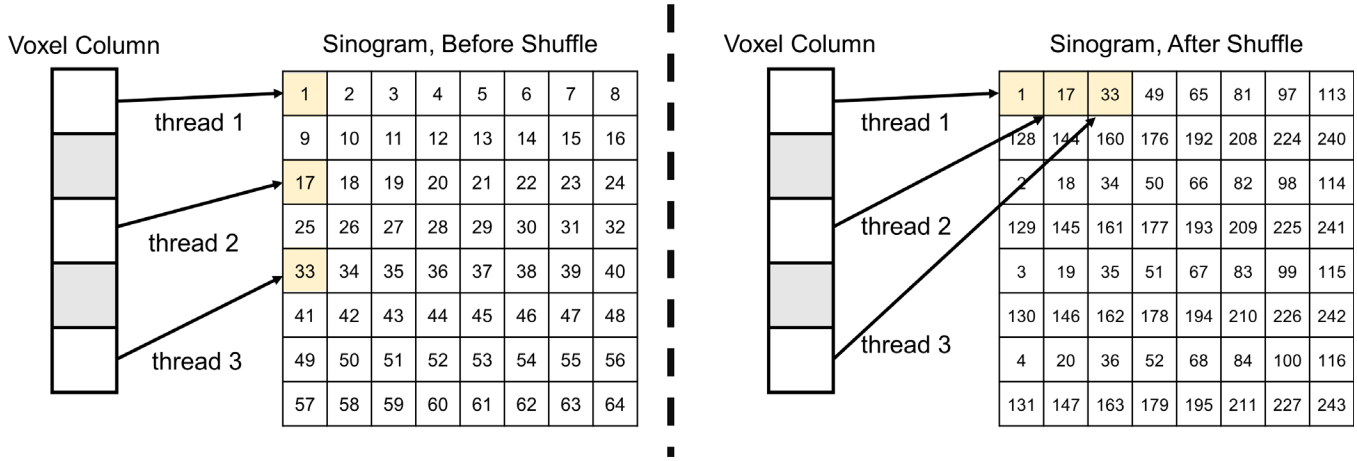


FIG. 1. Memory shuffle technique on a hypothetical system with a small dataset. A column of voxels sharing the same (x, y) coordinates but different z-coordinates is selected, and CUDA threads operate on every other voxel. Skipped voxels are shaded in gray. Arrows point to the entries in the sinogram that must be accessed by the threads in order to perform projection. (Left) With conventional indexing, threads access sinogram elements at intervals of a fixed distance. If the fastest changing index is the view direction, this distance is $2N_{sep}$. We use $N_{sep} = 8$ in this figure for compactness. Not all voxels are shown. The numbers in each box represent the index of the voxel. (Right) With the memory shuffle, the sinogram entries are permuted such that threads access sequential sinogram elements, causing these memory accesses to coalesce into a single transaction.

parameters are shown in Table I. We structured our codes to match the reference code, FreeCT_ICD, and one of our primary goals was numerical agreement with the reference code when both programs were run to convergence. The current release of FreeCT_ICD does not implement statistical data weights, so we did not include them in our codes either. We used a quadratic regularizer in this work, although FreeCT_ICD also supports other options. We used a variant of Joseph's method for the system matrix[22] and assumed that these data were precomputed and already loaded into RAM for timing purposes. Generation of the system matrix was done using FreeCT_ICD and was cached to disk. The average number of entries per voxel in the SSM is about six per view. In a voxel-driven backprojector where the center of each voxel is projected onto the detector and bilinear interpolation is applied, we expect four entries per view in all cases. Conceptually, the variant of Joseph's method that we are using is similar to the distance-driven[28] or separable footprints[29] projectors in that voxels with large magnification factors (close to the x-ray source) are more involved. For example, a 1 mm voxel close to the source may present a 2.5 by 2.5 mm footprint on the detector, which (assuming 1 mm detector pixels) would require nine SSM entries in that view. See Hahn et al. (38) for a recent discussion on forward projectors.

We initialize the volume to an FBP reconstruction and update the voxels according to a schedule that visits each voxel in a random order, but visits all voxels once in each iteration.[1] In contrast to the previously released FreeCT, released in C++, our code was implemented in MATLAB to handle data I/O and problem setup. The GPU codes were written as a MEX file that was compiled with CUDA libraries.

Like in all MBIR methods, there are nonphysical artifacts in the outermost slices because the data are incomplete for these slices. When evaluating the quantitative RMS difference between two volumes, we discard the outer 32 slices on each side.

2.F. Theoretical performance limit

The computational performance of MBIR can be limited by its code architecture and the details of its implementation. A poorly written implementation may introduce bottlenecks. We compare the run time of our implementation to predictions from the memory bandwidth specification. We used a single NVIDIA GTX 1080, which has a memory bandwidth of 320 GB/s that is calculated from its memory clock rate of 10 GHz and a 256 bit wide memory interface. We emphasize that the memory bandwidth is derived directly from the hardware specification of the memory unit, and any program (even direct memory copy) cannot achieve 100% of the memory bandwidth due to overhead.

From Table I, we calculate that in each iteration, a total of $512 \times 512 \times 128 \times 6952$ entries must be read from the sinogram during backprojection. Considering the size of the single-precision float (4 bytes) and the memory bandwidth of 320 GB/s, an ideal backprojector limited only by memory bandwidth would require 2.9 s. The forward projector, which uses both a read and write, would require 5.8 s. However, these numbers ignore memory alignment. In most cases, the requested memory accesses will not be aligned. A float is fetched for every other slice, leading to a total request of 256 contiguous bytes. If the memory address of the first byte is an integer multiple of 128 bytes, the request is aligned and can coalesce into two 128-byte transactions. In other cases, a third transaction is needed. Hence, a more realistic bound for the forward projector and backprojector speed will be 50% slower at 4.4 and 8.7 s, respectively.

3. RESULTS

3.A. Agreement with reference code

Figure 2 shows a comparison between our code and the reference FreeCT_ICD when both codes are run with 400 iterations. FreeCT_ICD uses a lexicographical update scheme and our code uses a random update scheme, so it is not true that the two codes should match to within numerical precision. Instead, we expect differences to appear because of the differences in update order as well as the approximation of the simultaneous update of $N_S = 16$ in-plane voxels. The RMS difference (RMSD) between the two after 400 iterations each is 1.8 HU. Figure 3 shows a histogram of these difference values, showing a very strong peak at 0 HU with long, symmetric tails in each direction. The accelerated GPU ICD code required 133 min for 400 iterations, compared to 256 h for FreeCT_ICD, yielding a speedup factor of 115×.

FIG. 2. Comparison of (left) FreeCT ICD reconstruction with 400 iterations, (middle) accelerated GPU ICD reconstruction with 400 iterations and $N_S = 16$, and (right) difference image on a pediatric thorax dataset. Reconstructions are with (WL, WW) = (0, 800) HU. Difference image is with (WL, WW) = (0, 50) HU.

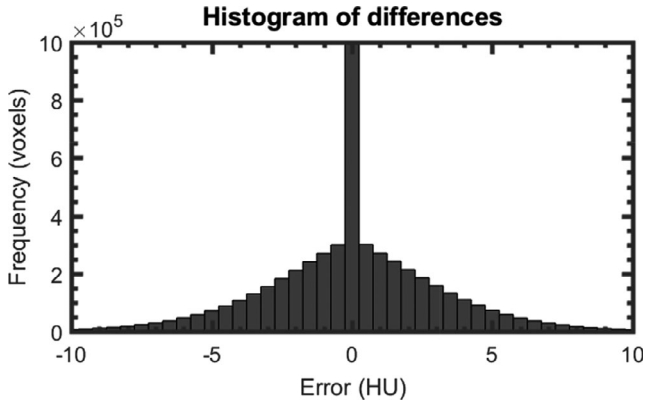


FIG. 3. Histogram of the difference between FreeCT ICD and the accelerated GPU ICD at 400 iterations. The total number of voxels in this histogram is 1.65×10^7. The bin at 0 HU extends to 1.25×10^7 and represents 75% of the voxels in the volume. Only 8% of the bar height is shown, with the remaining height truncated, in order to allow visualization of the other bars.

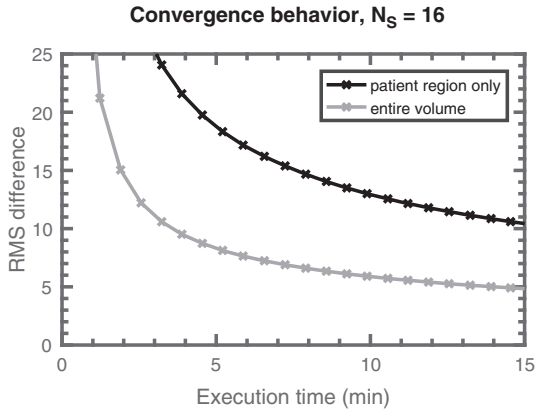


FIG. 4. Convergence of the accelerated GPU ICD code as a function of wall time with $N_S = 16$. Each marker corresponds to two iterations, with the first visible marker corresponding to four iterations.

Figure 4 shows the RMSD between GPU ICD and the 400 iteration FreeCT_ICD reference reconstruction as a function of iteration count. We used $N_S = 16$ in these plots. Our timing results do not include initialization steps, including reading the raw data and SSM, shuffling the sinogram, and transferring this data onto the GPU. This overhead currently requires about 5 min and has not been optimized. At 4 min, the code has processed 12 iterations with an RMSD of 9.5 HU. Reaching convergence to within 5 HU RMSD requires substantially longer, about 42 iterations or 14 min. It can be seen in Figure 2 that the RMSD is larger in the patient body compared to the surrounding air. Figure 4 also shows the RMSD in the patient body only. Reaching 10 HU RMSD within the patient body takes many more iterations.

3.B. Impact of multiple simultaneous update

Table II and Figure 5 summarizes the results for changing the N_S parameter. As expected, we found that the speed depends strongly on N_S for small N_S because more SMs can be recruited to work in parallel. Above $N_S = 16$, however, the performance plateaus as all SMs are at capacity. At $N_S = 1024$, divergence occurs at hundreds of iterations although the RMSD at 20 iterations or less is not much worse than $N_S = 1$. At $N_S = 4096$, divergence occurs immediately with rapidly escalating RMSD values. The initial RMSD between the FBP starting guess and the FreeCT reference reconstruction at 400 iterations is 51.7 HU.

TABLE II. RMS difference in HU between our accelerated GPU ICD code and the nearly converged reference FreeCT_ICD code at 400 iterations. Convergence is maintained until $N_S > 1024$.

	RMSD, 5 iterations	RMSD, 20 iterations	RMSD, 400 iterations
$N_S = 1$	17.4	7.23	1.77
$N_S = 4$	17.4	7.24	1.77
$N_S = 16$	17.4	7.24	1.77
$N_S = 64$	17.5	7.26	1.79
$N_S = 256$	17.5	7.35	1.85
$N_S = 1024$	18.1	7.93	4704
$N_S = 4096$	910	9350	–

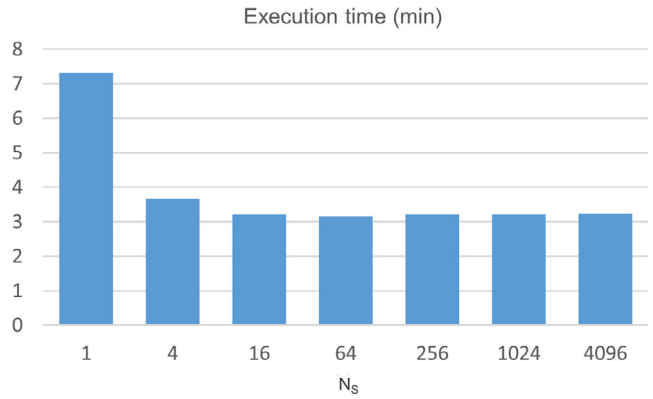


FIG. 5. Execution times for 10 iterations of GPU ICD for various values of N_S.

3.C. Code profiling

To understand the computational limitations of our code, we profiled the four steps of our algorithm in Table III with $N_S = 16$. These times were estimated by running variations of the accelerated GPU ICD code with certain kernels omitted and averaging execution times over 10 iterations. The first iteration, which does not include backprojection kernels or optimization kernels, was omitted from these timing results. The backprojection and forward projection require 11% and 41% more time than an idealized kernel that is subject only to memory bandwidth and the penalty for misaligned data access.

3.D. Sensitivity analysis

To assess the possible dependence of the code on acquisition factors such as helical pitch, source-to-isocenter distance,

TABLE III. Amount of time spent in each step of the accelerated GPU ICD code. The "bound" column refers to an ideal program that is limited only by the nominal memory bandwidth of the GPU with the additional penalty for misaligned data access, as described in Section 2.F.

Step	Time per iter (s)	Bound (s)
SSM copy	2.0	–
Backprojection kernel	4.9	4.4
Optimization kernel	0.6	–
Forward projection kernel	12.3	8.7
Total	19.7	13.1

SSM, stored system matrix.

and number of detector rows, we reprojected the converged FreeCT volume under a new acquisition geometry to create synthetic data. We then reconstructed this synthetic data using the initial FBP volume as a starting guess. The purpose of these experiments were to understand if variations in acquisition protocol would cause the optimization to slow down, not to assess convergence properties, which would be unrealistic given the nature of synthetic data. Table IV shows these reconstruction times. The run time for each iteration scales with the size of the SSM. Reducing the helical pitch from 1.0 to 0.6, for example, increases the SSM size by 67%, leading to an increase in time per iteration of 61%.

4. DISCUSSION

We have demonstrated an adaptation of MBIR using ICD for GPUs, achieving a reconstruction time of approximately 4 min for a 512 × 512 × 128 dataset and converging within 10 HU RMSD. Alongside other recent literature,[18–20] this work challenges the longstanding assumption that ICD for CT image reconstruction is not compatible with parallel computing platforms such as the GPU. In contrast to other existing work, we demonstrated our ICD code on a diagnostic helical CT dataset. Sabne et al. implemented ICD for a two-dimensional parallel-beam scan only.[18] They used thread-blocks that operated on overlapping supervoxels to obtain parallelism. The optimizations necessary in two dimensions are very different from three dimensions. On one hand, the third dimension provides a very natural avenue for parallelism. On the other hand, two-dimensional data have a greater amount of memory reuse and the L2 cache can be used to reduce memory reads from RAM. Sabne et al. also updated multiple voxels simultaneously but did not analyze convergence effects. Ha and Mueller,[20] in comparison, applied ICD to update multiple voxels in the z-direction only, as has been suggested by previous authors.[31] They demonstrated their algorithm on an axial CBCT scan only. They did not attempt to update multiple in-plane voxels simultaneously due to reasons of convergence. They ordered sinogram data so that the row direction was the fastest changing index to improve memory access. This works well for axial scans but is expected to achieve lower efficiency for diagnostic, helical CT.

TABLE IV. Sensitivity of run time on acquisition protocol factors. We used $N_S = 16$ in all these experiments. The code was run for 20 iterations. SID is source-to-isocenter distance. SSM size is the number of entries in the stored system matrix for the average reconstructed voxel.

Change	Pitch	Rows	SID (cm)	SSM size	Time per iter (s)
Baseline	1.0	16	59.5	6952	19.9
Reduced Pitch	0.6	16	59.5	11594	32.1
Reduced SID	1.0	16	49.5	7498	22.5
32-slice	1.0	32	59.5	6453	19.0

SSM, stored system matrix.

The optimization providing the largest speedup in our code was to reorder the sinogram so that memory accesses coalesce. A similar optimization has been used in other backprojection work.[25] A second optimization was the simultaneous update of multiple voxel columns, which breaks convergence guarantees that have traditionally been present in ICD. However, the ordered subsets technique[15] is used extensively in simultaneous update MBIR and is known to converge at best to a limit cycle, not the true minimum of the objective function. Also, MBIR is not run to convergence in clinical practice but terminated when the reconstruction is of sufficient quality. We find that a wide range of N_S values are available that achieve both sufficient parallelization and good convergence properties. On this dataset, any N_S value between 16 and 256 saturated the GPU while having minimal impact on convergence.

The run times reported here can be compared with other published work in MBIR. However, unless the dataset, target image, and system matrix are standardized, a fair comparison cannot be made. This standardization effort has occurred in the RabbitCT challenge for cone beam CT,[33] but no equivalent exists for MBIR in diagnostic CT geometries. Therefore, only rough comparisons can be made. In this work, we reconstructed a 736 × 16 × 13860 dataset to 512 × 512 × 128 with 10 HU RMSD in 4 min. Kim et al., in demonstrating ordered subsets with momentum, reconstructed a 888 × 64 × 2934 dataset to 512 × 512 × 154 with 10 HU RMSD in 4 min.[16] McGaffin and Fessler used a single GPU to reconstruct an 888 × 32 × 6852 dataset to a 512 × 512 × 109 reconstruction in 10 HU RMSD within 3 min.[17] We reiterate that exact comparisons cannot be made across these different datasets, but it can nonetheless be seen that our computational performance is comparable with existing simultaneous update approaches. In particular, our current codes do not use statistical weights or more complex regularizer functions[1] that may be costly to compute.

Statistical weights may especially affect convergence rates. Statistical weights were not included in our code because they are not yet featured in our reference FreeCT ICD code. The weights used in the original MBIR development[1] and in accelerated versions[16] are not simply the inverse of the variance and have not been disclosed, which complicates comparisons. Intuitively, one may expect that rays with small weights will be slow to propagate their information into the

reconstruction compared to rays with large weights. The inclusion of highly heterogeneous statistical weights may reduce the maximum allowed N_S value prior to convergence. Other geometries may also decrease the maximum allowed N_S. However, even if the maximum allowed N_S would be to reduced by an order of magnitude from 256 to 25, it would still be above our choice of $N_S = 16$ which saturates the GPU.

As shown in Section 3.C, there is limited room for further improvement in our execution times. The SSM copy times can be hidden using CUDA streams. The backprojection kernel is close to its theoretical bound when considering the impact of memory alignment. The forward projector is slower, possibly due to the extensive use of atomic updates. With a precomputed voxel update schedule, it may be possible to eliminate atomic writes by decomposing a forward projection step into two or more discrete kernel calls, with the SSM structured such that a detector pixel is guaranteed to be updated once per kernel call. The maximum improvement possible, however, is small because the forward projector is also within 40% of its theoretical bound. This suggests that drastic improvements in performance requires re-architecture of the underlying algorithm. One such option is block-based ICD,[34] which uses memory more efficiently to update multiple voxels. Another option that has been exploited on the CPU is NU-ICD, which updates voxels according to expected improvement in cost function.[32] It would be more challenging to implement NU-ICD with our current approach because we update columns of voxels in the z-direction simultaneously.

Our algorithm inherits some limitations from the current FreeCT_ICD reference code. The direct output of this code is in a rotating frame of reference and requires derotation prior to storage, leading to possible resolution loss, but encouraging results in this aspect were previously shown.[22] As with FreeCT_ICD, run times are proportional to the SSM size. The SSM size is inversely proportional to the pitch. Including focal spot wobble in either the z- or x- directions would similarly increase the SSM size. We did not use statistical weights, and we use a variation in Joseph's projector. The use of a more sophisticated projector[29] could potentially improve reconstruction characteristics, but data on this question are conflicting.[35,36] Conversely, a memory efficient projector would increase speed but would also approximate detector physics.[27] Wang et al. similarly found that using an approximate projector model allowed for much higher speed.[37] Note that the intention of the memory efficient system matrix proposed by Guo and Gao[27] was to allow the SSM to reside on the GPU. In our architecture, the SSM resides in CPU RAM and is moved in chunks to the GPU during each iteration. A smaller SSM would directly translate to reduced memory bandwidth requirements.

In summary, we have demonstrated that ICD for MBIR can be adapted onto GPUs for reconstructing diagnostic, helical CT datasets. These are interim results of the FreeCT project,[22,23] and in future work, we plan to validate these codes against product reconstructions on clinical datasets and to release them publicly for academic research.

CONFLICT OF INTEREST

The authors have no conflict of interest to disclose.

[a]Author to whom correspondence should be addressed. Electronic mail: shsieh@mednet.ucla.edu.

REFERENCES

1. Thibault JB, Sauer KD, Bouman CA, Hsieh J. A three-dimensional statistical approach to improved image quality for multislice helical CT. *Med Phys*. 2007;34:4526.
2. Kak A, Slaney M. *Principles of Computerized Tomographic Imaging*. Philadelphia, PA: SIAM; 1988.
3. Pan X, Sidky EY, Vannier M. Why do commercial CT scanners still employ traditional, filtered back-projection for image reconstruction? *Inverse Prob*. 2009;25:123009.
4. Mileto A, Zamora DA, Alessio AM, et al. CT detectability of small low-contrast hypoattenuating focal lesions. Iterative reconstructions versus filtered back projection. *Radiology*. 2018;289:443–454.
5. Singh S, Kalra MK, Do S, et al. Comparison of hybrid and pure iterative reconstruction techniques with conventional filtered back projection: dose reduction potential in the abdomen. *J Comput Assist Tomogr*. 2012;36:347–353.
6. Kalra MK, Naz N, Rizzo SMR, Blake MA. Computed tomography radiation dose optimization: scanning protocols and clinical applications of automatic exposure control. *Curr Probl Diagn Radiol*. 2005;34:171–181.
7. Vaishnav J, Jung W, Popescu L, Zeng R, Myers K. Objective assessment of image quality and dose reduction in CT iterative reconstruction. *Med Phys*. 2014;41:071904.
8. Pickhardt PJ, Lubner MG, Kim DH, et al. Abdominal CT with model-based iterative reconstruction (MBIR): initial results of a prospective trial comparing ultralow-dose with standard-dose imaging. *AJR Am J Roentgenol*. 2012;199:1266–1274.
9. Pooler BD, Lubner MG, Kim DH, et al. Prospective trial of the detection of urolithiasis on ultralow dose (sub mSv) noncontrast computerized tomography: direct comparison against routine low dose reference standard. *J Urol*. 2014;192:1433–1439.
10. Hsieh SS, Chesler DA, Fleischmann D, Pelc NJ. A limit on dose reduction possible with CT reconstruction algorithms without prior knowledge of the scan subject. *Med Phys*. 2016;43:1361–1368.
11. Kachelrieß M, Watzke O, Kalender WA. Generalized multi-dimensional adaptive filtering for conventional and spiral single-slice, multi-slice, and cone-beam CT. *Med Phys*. 2001;28:475–490.
12. Zheng X, Ravishankar S, Long Y, Fessler JA. PWLS-ULTRA: an efficient clustering and learning-based approach for low-dose 3D CT image reconstruction. *IEEE Trans Med Imaging*. 2018;37:1498–1510.
13. Li Z, Yu L, Trzasko JD, et al. Adaptive nonlocal means filtering based on local noise level for CT denoising. *Med Phys*. 2014;41:011908.
14. Chen H, Zhang Y, Kalra MK, et al. Low-dose CT with a residual encoder-decoder convolutional neural network. *IEEE Trans Med Imaging*. 2017;36:2524–2535.
15. Erdogan H, Fessler JA. Ordered subsets algorithms for transmission tomography. *Phys Med Biol*. 1999;44:2835.
16. Kim D, Ramani S, Fessler JA. Combining ordered subsets and momentum for accelerated X-ray CT image reconstruction. *IEEE Trans Med Imaging*. 2015;34:167–178.
17. McGaffin MG, Fessler JA. Alternating dual updates algorithm for X-ray CT reconstruction on the GPU. *IEEE Transactions on Computational Imaging*. 2015;1:186–199.
18. Sabne A, Wang X, Kisner SJ, Bouman CA, Raghunathan A, Midkiff SP. Model-based iterative CT image reconstruction on GPUs. *ACM SIGPLAN Notices*. 2017;52:207–220.
19. Li X, Liang Y, Zhang W, et al. In: cuMBIR: An efficient framework for low-dose X-ray CT image reconstruction on GPUs. Proceedings of the 2018 international conference on supercomputing; ACM; 2018. p. 184–194.

20. Ha S, Mueller K. A GPU-accelerated multi-voxel update scheme for iterative coordinate descent (ICD) optimization in statistical iterative CT reconstruction (SIR). *IEEE Transactions on Computational Imaging*. 2018;4:35–365.
21. Chen B, Duan X, Yu Z, Leng S, Yu L, McCollough C. Development and validation of an open data format for CT projection data. *Med Phys*. 2015;42:6964–6972.
22. Hoffman JM, Noo F, Young S, Hsieh SS, McNitt-Gray M. Free CT_ICD: an open source implementation of a Model-Based iterative reconstruction method using coordinate descent optimization for CT imaging investigations. *Med Phys*. 2018;45:3591–3603.
23. Hoffman J, Young S, Noo F, McNitt-Gray M. FreeCT_wFBP: a robust, efficient, open-source implementation of weighted filtered backprojection for helical, fan-beam CT. *Med Phys*. 2016;43:1411–1420.
24. Xu J, Tsui BM. Iterative image reconstruction in helical cone-beam x-ray CT using a stored system matrix approach. *Phys Med Biol*. 2012;57:3477.
25. Steckmann S, Knaup M, Kachelrieß M. Algorithm for hyperfast cone-beam spiral backprojection. *Comput Methods Programs Biomed*. 2010;98:253–260.
26. Steckmann S, Knaup M, Kachelrieß M. High performance cone-beam spiral backprojection with voxel-specific weighting. *Phys Med Biol*. 2009;54:3691.
27. Guo M, Gao H. Memory-efficient algorithm for stored projection and backprojection matrix in helical CT. *Med Phys*. 2017;44:1287–1300.
28. De Man B, Basu S. Distance-driven projection and backprojection in three dimensions. *Phys Med Biol*. 2004;49:2463.
29. Long Y, Fessler JA, Balter JM. 3D forward and back-projection for X-ray CT using separable footprints. *Med Imaging IEEE Trans*. 2010;29:1839–1850.
30. Boyd SP, Vandenberghe L. *Convex Optimization*. Cambridge, UK: Cambridge university press; 2004.
31. Fessler JA, Kim D. In: Axial block coordinate descent (ABCD) algorithm for X-ray CT image reconstruction. 11th international meeting on fully three-dimensional image reconstruction in radiology and nuclear medicine; Citeseer; 2011.
32. Yu Z, Thibault J, Bouman CA, Sauer KD, Hsieh J. Fast model-based X-ray CT reconstruction using spatially nonhomogeneous ICD optimization. *IEEE Trans Image Process*. 2011;20:161–175.
33. Rohkohl C, Keck B, Hofmann H, Hornegger J. RabbitCT—an open platform for benchmarking 3D cone-beam reconstruction algorithms. *Med Phys*. 2009;36:3940–3944.
34. Benson TM, De Man BK, Fu L, Thibault J. In: Block-based iterative coordinate descent. Nuclear science symposium conference record (NSS/MIC), 2010 IEEE; IEEE; 2010. p. 2856–2859.
35. Hofmann C, Knaup M, Kachelrieß M. Effects of ray profile modeling on resolution recovery in clinical CT. *Med Phys*. 2014; 41(2):021907.
36. Do S, Karl WC, Singh S, et al. High fidelity system modeling for high quality image reconstruction in clinical CT. *PLoS ONE*. 2014;9:e111625.
37. Wang AS, Stayman JW, Otake Y, Vogt S, Kleinszig G, Siewerdsen JH. Accelerated statistical reconstruction for c-arm cone-beam CT using nesterov's method. *Med Phys*. 2015;42:2699–2708.
38. Hahn K, Schondube H, Stierstorfer K, Hornegger J, Noo F. A comparison of linear interpolation models for iterative CT reconstruction. *Med Phys*. 2016;42:6455–6473.