Alternating dual updates algorithm for X-ray CT reconstruction on the GPU

Madison G. McGaffin, *Member, IEEE,* and Jeffrey A. Fessler, *Fellow, IEEE*

***Abstract*—Model-based image reconstruction (MBIR) for X-ray computed tomography (CT) offers improved image quality and potential low-dose operation, but has yet to reach ubiquity in the clinic. MBIR methods form an image by solving a large statistically motivated optimization problem, and the long time it takes to numerically solve this problem has hampered MBIR's adoption. We present a new optimization algorithm for X-ray CT MBIR based on duality and group coordinate ascent that may converge even with approximate updates and can handle a wide range of regularizers, including total variation (TV). The algorithm iteratively updates groups of dual variables corresponding to terms in the cost function; these updates are highly parallel and map well onto the GPU. Although the algorithm stores a large number of variables, the "working size" for each of the algorithm's steps is small and can be efficiently streamed to the GPU while other calculations are being performed. The proposed algorithm converges rapidly on both real and simulated data and shows promising parallelization over multiple devices.**

I. Introduction

X-ray computed tomography (CT) model-based image reconstruction (MBIR) combines information about system physics, measurement statistics, and prior knowledge about images into a high-dimensional cost function [1]. The variate of this function is an image; the image that minimizes this cost function can contain less noise and fewer artifacts than those produced with conventional analytical techniques, especially at reduced doses [1]–[3].

The primary drawback of MBIR methods is how long it takes to find this minimizer. In addition to general optimization algorithms like conjugate gradients with specialized preconditioners [4], [5], a wide range of CT-specialized algorithms have been proposed to accelerate the optimization. One popular approach uses iterated coordinate descent (ICD) to sequentially update pixels (or groups of pixels) of the image [6], [7]. Since it is a sequential algorithm, ICD faces challenges from stagnating processor clock speeds and cannot exploit the increasing parallelization in modern computing hardware.

Another family of algorithms uses variable splitting and alternating minimization techniques to separate challenging parts of the cost function into more easily solved subproblems [8]–[11]. When used with the ordered subsets (OS) approximation [10], [12], these algorithms can converge very rapidly. Unfortunately, without relaxation, OS-based algorithms have uncertain convergence properties. Nonetheless, combining OS with accelerated first-order methods [13], [14] has produced simple algorithms with state of the art convergence speeds.

Supported in part by NIH grant U01 EB 018753, and by equipment donations from Intel Corporation. The authors are with the EECS Department, University of Michigan, Ann Arbor, MI, 48109-2122. Email: {mcgaffin,fessler}@umich.edu.

This paper proposes an algorithm that shares some properties with prior works. Like some variable splitting methods, our proposed algorithm consists of steps that consider parts of the cost function in isolation. Separating jointly challenging parts of the cost function from one another allows us to use specialized and fast solvers for each part. Our algorithm also uses a group coordinate optimization scheme, somewhat like ICD, but the variables it updates are in a dual domain; updating a small group of dual variables can simultaneously update many image pixels. Like OS algorithms, our algorithm need not visit all the measured data to update the image, but unlike OS algorithms without relaxation, the proposed algorithm has some convergence guarantees.

The next section sets up our MBIR CT reconstruction problem. Section II introduces the mathematics of the proposed algorithm, and Section III describes our single- and multiple-device implementations. Section IV provides some experimental results and Section V gives some conclusions and directions for future work.

A. Model-based image reconstruction

Consider the following X-ray CT reconstruction problem [1]:

$$\widehat{\mathbf{x}} \in \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmin}} \ \mathsf{L}(\mathbf{A}\mathbf{x}) + \mathsf{R}(\mathbf{C}\mathbf{x}). \tag{1}$$

There are M measurements and N pixels, and we constrain all the pixels of the image $\mathbf{x} \in \mathbb{R}^N$ to be nonnegative. The CT projection matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ models system physics and geometry, and the finite differencing matrix $\mathbf{C} \in \mathbb{R}^{K \times N}$ computes the differences between each pixel and its neighbors. The number of differences penalized in the image, K, is a multiple of N. For example, penalizing differences along the three cardinal 3D axes would set $K = 3N$. The matrices $\mathbf{A}$ and $\mathbf{C}$ are too large to store in memory, so multiplication with these matrices and their adjoints is implemented "on the fly".

Both L and R are separable sums of convex functions:

$$\mathsf{L}(\mathbf{p}) = \sum_{i=1}^{M} l_i(p_i), \qquad \mathsf{R}(\mathbf{d}) = \sum_{k=1}^{K} r_k(d_k). \tag{2}$$

We call L the data-fit term because it penalizes discrepancies between the measured data $\mathbf{y} \in \mathbb{R}^M$ and the CT projection of $\mathbf{x}$. A common choice for L is a weighted sum of quadratic functions; *i.e.*,

$$l_i(p_i) = \frac{w_i}{2}(p_i - y_i)^2, \tag{3}$$

with the weight $w_i > 0$. Traditionally, the weight is the inverse of the variance of the ith measurement, $w_i = 1/\sigma_i^2$.

Similarly, R encourages regularity of the reconstructed image $\widehat{\mathbf{x}}$ by penalizing the differences between neighboring pixels. R is a weighted sum of penalized differences,

$$r_k(d_k) = \beta_k \psi(d_k). \tag{4}$$

The potential function ψ is convex, even, usually nonquadratic, and coercive. The quadratic penalty function, $\psi(t) = \frac{1}{2}t^2$, while analytically tractable, tends to favor reconstructed images $\widehat{\mathbf{x}}$ with blurry edges because it penalizes large differences between neighboring pixels (*i.e.*, edges) aggressively. Potential functions $\psi(t)$ that have a smaller rate of growth as $|t| \to \infty$ are called edge-preserving because they penalize these large differences less aggressively. Examples include the absolute value function $\psi(t) = |t|$ from total variation (TV) regularization, the Fair potential, and the q-generalized Gaussian. The positive weights $\{\beta_k\}$ are fixed and encourage certain resolution or noise properties in the image [15], [16].

The functions L and R have opposing effects on the reconstructed image $\widehat{\mathbf{x}}$: L encourages data fidelity and can lift the noise from the data $\mathbf{y}$ into the reconstructed image, whereas R encourages smoothness at the cost of producing an image $\mathbf{x}$ that does not fit the measurements as well. Combining L and R complicates the task of finding a minimizer $\widehat{\mathbf{x}}$. Without the regularizer R, the reconstruction problem (1) could possibly be solved using a fast quadratic solver. Conversely, without the data-fit term L (without CT system matrix $\mathbf{A}$), (1) becomes a denoising problem for which many fast algorithms exist, including methods suitable for the GPU [17].

Variable splitting and alternating minimization provide a framework for separating L and R into different subproblems [18]. The ADMM algorithm in [8] used a circulant approximation to the Gram matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ to provide rapid convergence rates for 2D CT problems. Unfortunately, the circulant approximation is less useful in 3D CT. We partially overcame these difficulties in [19] by using a duality-based approach to solving problems involving the CT system matrix, but the resulting algorithm still used ADMM, which has difficult-to-tune penalty parameters and relatively high memory use. Gradient-based algorithms like OS [12] with acceleration [13] and the linearized augmented Lagrangian method with ordered subsets [10] (OS-LALM), can produce rapid convergence rates but use an approximation to the gradient of the data-fit term and have uncertain convergence properties. Some of these algorithms require generalizations to handle non-smooth regularizers.

This paper describes an extension of the algorithms in [19], [20]. The proposed algorithm uses duality, group coordinate ascent with carefully chosen groups, and the majorize-minimize framework to rapidly solve the reconstruction problem (1). We extend [20] by also considering the nonnegativity constraint $\mathbf{x} \geq \mathbf{0}$ in (1). Our algorithm is designed for the GPU: while it uses many variables, the "working set" for each of the algorithm's steps is small and easily fits in GPU memory. We stream these groups of variables to the GPU and hide the latency of these transfers by performing other, less memory-intensive computations. We show that the proposed algorithm can be implemented with multiple GPUs for additional acceleration.

II. Reconstruction algorithm

At a high level, the proposed algorithm approximately performs the following iteration:

$$\mathbf{x}^{(n+1)} = \operatorname*{argmin}_{\mathbf{x}} \mathsf{J}^{(n)}(\mathbf{x}), \qquad \text{where} \tag{5}$$

$$\mathsf{J}^{(n)}(\mathbf{x}) = \mathsf{L}(\mathbf{A}\mathbf{x}) + \mathsf{R}(\mathbf{C}\mathbf{x}) + \mathsf{N}(\mathbf{x}) + \frac{\mu}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|^2, \tag{6}$$

with $\mu > 0$. We have expressed the nonnegativity constraint $\mathbf{x} \geq \mathbf{0}$ as the characteristic function N:

$$\mathsf{N}(\mathbf{x}) = \sum_{j=1}^{N} \iota_j(x_j), \qquad \text{where} \tag{7}$$

$$\iota_k(x) = \begin{cases} 0, & x \geq 0; \\ \infty, & \text{else.} \end{cases} \tag{8}$$

Although ι_k is discontinuous, it is convex. Iteratively solving (5) exactly would produce a sequence $\left\{\mathbf{x}^{(n)}\right\}$ that converges to a minimizer $\widehat{\mathbf{x}}$ regardless of the choice of μ.

The function $\mathsf{J}^{(n)}(\mathbf{x})$ appears to be as difficult to minimize as the original cost function (1). Even if $\mathsf{J}^{(n)}(\mathbf{x})$ could be minimized exactly, $\mathbf{x}^{(n+1)}$ is likely not a solution to the original reconstruction problem (1). However, the additional proximal term provides structure that allows us to approximately solve $\mathsf{J}^{(n)}$ efficiently using the following duality-based technique. The parameter μ controls the tradeoff between how efficiently we can approximately minimize $\mathsf{J}^{(n)}$ and how close the minimizer of $\mathsf{J}^{(n)}$ is to a minimizer of the original cost function (1).

Let l_i^*, r_k^* and ι_j^* denote the convex conjugates of l_i, r_k and ι_j, respectively, *e.g.*, :

$$l_i^*(u_i) = \sup_{p_i} p_i u_i - l_i(p_i). \tag{9}$$

Because l_i, r_k and ι_j are convex, they are equal to the convex conjugates of l_i^*, r_k^* and ι_j^*, respectively. We use this biconjugacy property to write

$$l_i(p_i) = \sup_{u_i} p_i u_i - l_i^*(u_i). \tag{10}$$

By summing over the indices i, j and k, we write L, R and N implicitly as the suprema of sums of one-dimensional dual functions:

$$\mathsf{L}(\mathbf{p}) = \sup_{\mathbf{u}} \sum_{i=1}^{M} p_i u_i - l_i^*(u_i) = \sup_{\mathbf{u}} \mathbf{u}^\mathsf{T}\mathbf{p} - \mathsf{L}^*(\mathbf{u}), \tag{11}$$

$$\mathsf{R}(\mathbf{d}) = \sup_{\mathbf{v}} \sum_{k=1}^{K} v_k d_k - r_k^*(d_k) = \sup_{\mathbf{v}} \mathbf{v}^\mathsf{T}\mathbf{d} - \mathsf{R}^*(\mathbf{v}), \tag{12}$$

$$\mathsf{N}(\mathbf{x}) = \sup_{\mathbf{z}} \sum_{j=1}^{N} z_j x_j - \iota_j^*(z_j) = \sup_{\mathbf{z}} \mathbf{z}^\mathsf{T}\mathbf{x} - \mathsf{N}^*(\mathbf{z}). \tag{13}$$

With (11)-(13), we rewrite the update problem (5) as

$$\mathbf{x}^{(n+1)} = \operatorname*{argmin}_{\mathbf{x}} \sup_{\mathbf{u},\,\mathbf{v},\,\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathbf{z}), \tag{14}$$

$$\mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}) \triangleq \frac{\mu}{2}\left|\left|\mathbf{x}-\mathbf{x}^{(n)}\right|\right|^2 + (\mathbf{A}^\intercal\mathbf{u}+\mathbf{C}^\intercal\mathbf{v}+\mathbf{z})^\intercal\mathbf{x} - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{N}^*(\mathbf{z}). \quad (15)$$

Reversing the order of minimization and maximization[1] yields:

$$\min_{\mathbf{x}} \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}) = \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \min_{\mathbf{x}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}). \quad (16)$$

The now inner minimization over $\mathbf{x}$ is trivial to perform. We solve for the minimizer $\mathbf{x}$ and write it in terms of the dual variables $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{z}$:

$$\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}) = \mathbf{x}^{(n)} - \frac{1}{\mu}(\mathbf{A}^\intercal\mathbf{u}+\mathbf{C}^\intercal\mathbf{v}+\mathbf{z}). \quad (17)$$

The image induced by the dual variables, $\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z})$, minimizes the update cost function (5) when $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{z}$ maximize the following dual function[2]:

$$\mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}) \triangleq \mathsf{S}^{(n)}\Big(\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}),\mathbf{u},\mathbf{v},\mathbf{z}\Big) \quad (18)$$

$$= -\frac{1}{2\mu}||\mathbf{A}^\intercal\mathbf{u}+\mathbf{C}^\intercal\mathbf{v}+\mathbf{z}||^2 + (\mathbf{A}^\intercal\mathbf{u}+\mathbf{C}^\intercal\mathbf{v}+\mathbf{z})^\intercal\mathbf{x}^{(n)} - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{N}^*(\mathbf{z}). \quad (19)$$

Maximizing (19), *i.e.*, solving the dual problem (16), induces an image (17) that minimizes the update problem (5). We maximize (19) approximately using a stochastic group coordinate ascent algorithm described in the next section. Under conditions similar to other alternating minimization algorithms like ADMM [21], the proposed algorithm may converge even with these approximate updates; see Appendix C.

At a high level, our proposed algorithm iteratively performs the following steps:

1) form the dual function $\mathsf{D}^{(n)}$ in (19) using $\mathbf{x}^{(n)}$,
2) find $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ by running iterations of the SGCA algorithm detailed in the following sections:

$$\mathbf{u}^{(n+1)},\mathbf{v}^{(n+1)},\mathbf{z}^{(n+1)} \approx \operatorname*{argmax}_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}), \quad (20)$$

3) and update $\mathbf{x}^{(n+1)}$:

$$\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\Big(\mathbf{u}^{(n+1)},\mathbf{v}^{(n+1)},\mathbf{z}^{(n+1)}\Big). \quad (21)$$

A. Stochastic group coordinate ascent

We propose an SGCA algorithm to perform the dual maximization (20). The algorithm iteratively selects a group of variables (in our case, a set of the elements of the dual variables $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{z}$) via a random process and updates them to increase the value of the dual function $\mathsf{D}^{(n)}$. Because SGCA is convergent [22], enough iterations of the algorithm in this section will produce dual solutions $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ that are arbitrarily close to true maximizers $\widehat{\mathbf{u}}^{(n+1)}$, $\widehat{\mathbf{v}}^{(n+1)}$ and $\widehat{\mathbf{z}}^{(n+1)}$. However, the solution accuracy of more interest is how well the induced image $\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\big(\mathbf{u}^{(n+1)},\mathbf{v}^{(n+1)},\mathbf{z}^{(n+1)}\big)$ approximates the exact minimizer of (5). The data-fit and regularizer dual variables $\mathbf{u}$ and $\mathbf{v}$ affect the induced image $\widetilde{\mathbf{x}}^{(n+1)}$, per (17), through the linear operators $\mathbf{A}^\intercal$ and $\mathbf{C}^\intercal$ respectively. These linear operators propagate the influence of a possibly small group of dual variables to many pixels: *e.g.*, the elements of $\mathbf{u}$ corresponding to a single projection view are backprojected over a large portion of the image. Consequently, performing a just a few dual group updates can significantly improve the image $\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z})$.

An SGCA algorithm updates one group of variables at a time. We can form these groups arbitrarily, and as long as each group is visited "often enough" the algorithm converges to a solution [22]. To exploit the structure of $\mathsf{D}^{(n)}$, we choose each group so that it contains elements from only $\mathbf{u}$, $\mathbf{v}$ or $\mathbf{z}$; *i.e.*, no group contains elements from different variables. Sections II-B, II-C, and II-D describe the updates for each of these groups.

Because the SGCA algorithm updates elements of the dual variables in random order, conventional iteration notation becomes cumbersome. Instead, mirroring the algorithm's implementation, we describe the updates as occurring "in-place." The "new" value of a variable is written with a superscripted plus, *e.g.*, $\mathbf{u}^+$. To refer to the "current" value of a dual variable in an update problem, we use a superscripted minus; *e.g.*, $\mathbf{u}^-$. For example to update $\mathbf{u}$ to maximize $\mathsf{D}^{(n)}$ while holding the other variables constant, we write $\mathbf{u}^+ = \arg\max_{\mathbf{u}} \mathsf{D}^{(n)}(\mathbf{u},\mathbf{v}^-,\mathbf{z}^-)$. That is, we replace the contents of $\mathbf{u}$ in memory with the maximizing value $\mathbf{u}^+$.

We rewrite the quadratic and linear terms in $\mathsf{D}^{(n)}$ using this notation and (17) by rewriting the quadratic and linear terms in (19):

$$\mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}) = c - \frac{1}{2\mu}\left|\left|\mathbf{A}^\intercal\big(\mathbf{u}-\mathbf{u}^-\big) + \mathbf{C}^\intercal\big(\mathbf{v}-\mathbf{v}^-\big) + \mathbf{z} - \mathbf{z}^-\right|\right|^2 + (\mathbf{A}^\intercal\mathbf{u}+\mathbf{C}^\intercal\mathbf{v}+\mathbf{z})^\intercal\widetilde{\mathbf{x}} - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{N}^*(\mathbf{z}), \quad (22)$$

where the buffer $\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u}^-,\mathbf{v}^-,\mathbf{z}^-)$ and the constant $c = \mathsf{D}^{(n)}(\mathbf{u}^-,\mathbf{v}^-,\mathbf{z}^-) - (\mathbf{A}^\intercal\mathbf{u}^- + \mathbf{C}^\intercal\mathbf{v}^- + \mathbf{z}^-)^\intercal\widetilde{\mathbf{x}}$ is independent of $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{z}$. After any group of elements of the dual variables is updated, we update $\widetilde{\mathbf{x}}$ to reflect the changed dual variables (17). The following sections detail these dual variable updates.

B. Tomography ($\mathbf{u}$) updates

Consider maximizing (22) with respect to some subset of the elements of $\mathbf{u}$,

$$\mathbf{u}_g^+ = \operatorname*{argmax}_{\mathbf{u}_g} \mathsf{D}^{(n)}\big(\mathbf{u},\mathbf{v}^-,\mathbf{z}^-\big) \quad (23)$$

$$= \operatorname*{argmax}_{\mathbf{u}_g} -\frac{1}{2\mu}\left|\left|\mathbf{u}_g-\mathbf{u}_g^-\right|\right|^2_{\mathbf{A}_g\mathbf{A}_g^\intercal} - \mathsf{L}_g^*(\mathbf{u}_g) + \mathbf{u}_g^\intercal\mathbf{A}_g\widetilde{\mathbf{x}}, \quad (24)$$

where $\mathbf{u}_g$ is a subset of the elements of $\mathbf{u}$. The elements of $\mathbf{u}_g$ are coupled in (24) by the matrix $\mathbf{A}_g\mathbf{A}_g^\intercal$, where $\mathbf{A}_g$ contains the rows of $\mathbf{A}$ corresponding to the group $\mathbf{u}_g$.

If $\mathbf{A}_g\mathbf{A}_g^\intercal$ were diagonal, *i.e.*, if the rays corresponding to elements of $\mathbf{u}_g$ were nonoverlapping, then solving (24) would be trivial. (Of course this is also the case when $\mathbf{u}_g$ is a single

[1] See Appendix A.
[2] See Appendix B.

element of $\mathbf{u}$). However, updating $\mathbf{u}_g$ using only nonoverlapping rays would limit the algorithm's parallelizability. Existing CT projector software may also not be able to efficiently compute the projection and backprojection of individual rays instead of *e.g.*, a projection view at a time. If we allow $\mathbf{u}_g$ to contain overlapping rays, then the coupling induced by $\mathbf{A}_g\mathbf{A}_g^\intercal$ makes (24) expensive to solve exactly. Instead of pursuing the exact solution to (24) or using a line search with GPU-unfriendly inner products, we use a minorize-maximize technique [23], [24] to find an approximate solution that still increases the dual function $\mathsf{D}^{(n)}$.

Let the diagonal matrix $\mathbf{M}_g$ majorize $\mathbf{A}_g\mathbf{A}_g^\intercal$, *i.e.*, the matrix $\mathbf{M_g} - \mathbf{A}_g\mathbf{A}_g^\intercal$ has no negative eigenvalues. Solving the following separable problem produces an updated $\mathbf{u}_g^+$ that increases the dual function $\mathsf{D}^{(n)}$:

$$\mathbf{u}_g^+ = \underset{\mathbf{u}_g}{\operatorname{argmax}} \; -\frac{1}{2\mu}\left|\left|\mathbf{u}_g - \mathbf{u}_g^-\right|\right|_{\mathbf{M}_g}^2 - \mathsf{L}_g^*(\mathbf{u}_g) + \mathbf{u}_g^\intercal \mathbf{A}_g \widetilde{\mathbf{x}}. \tag{25}$$

In the common case that $\mathsf{L}(\mathbf{Ax}) = \frac{1}{2}||\mathbf{Ax}-\mathbf{y}||_{\mathbf{W}}^2$, *i.e.*, $l_i(p_i) = \frac{w_i}{2}(p_i - y_i)^2$, the conjugate L_g^* is

$$\mathsf{L}_g^*(\mathbf{u}_g) = \frac{1}{2}||\mathbf{u}_g||_{\mathbf{W}_g^{-1}}^2 + \mathbf{u}_g^\intercal \mathbf{y}_g, \tag{26}$$

and the solution to (25) is

$$\mathbf{u}_g^+ = (\mathbf{W}_g\mathbf{M}_g + \mu\mathbf{I})^{-1}\mathbf{W}_g\big(\mu(\mathbf{A}_g\widetilde{\mathbf{x}} - \mathbf{y}_g) + \mathbf{M}_g\mathbf{u}_g^-\big). \tag{27}$$

It is computationally challenging to find an "optimal" diagonal majorizing matrix $\mathbf{M}_g \succeq \mathbf{A}_g\mathbf{A}_g^\intercal$, but the following matrix majorizes $\mathbf{A}_g\mathbf{A}_g^\intercal$ and is easy to compute [12]:

$$\mathbf{M}_g = \operatorname*{diag}_i\Big\{\big[\mathbf{A}_g\mathbf{A}_g^\intercal\mathbf{1}\big]_i\Big\}. \tag{28}$$

This choice of $\mathbf{M}_g$ depends only on the system geometry through $\mathbf{A}_g$ and not on any patient-specific data. Provided the groups and geometry are determined beforehand, these majorizers can be precomputed. This was the case for our experiments, and we used one group per view. Storing the diagonals of all the majorizers $\{\mathbf{M}_g\}$ took the same amount of memory as the noisy projections $\mathbf{y}$.

After updating the group $\mathbf{u}_g$ (27), we "backproject" the update into the buffer $\widetilde{\mathbf{x}}$ (17):

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\mathbf{A}_g^\intercal\big(\mathbf{u}_g^+ - \mathbf{u}_g^-\big). \tag{29}$$

Altogether, updating $\mathbf{u}_g$ and $\widetilde{\mathbf{x}}$ requires a forward projection and backprojection for the rays in group g and a few vector operations (27).

Running one iteration of the minorize-maximize (MM) operation (27) will not exactly maximize $\mathsf{D}^{(n)}$ with respect to $\mathbf{u}_g$. One could run more MM iterations to further increase $\mathsf{D}^{(n)}$, but we have not found this necessary in our experiments. In contrast, the updates for the denoising and nonnegativity variables below are exact.

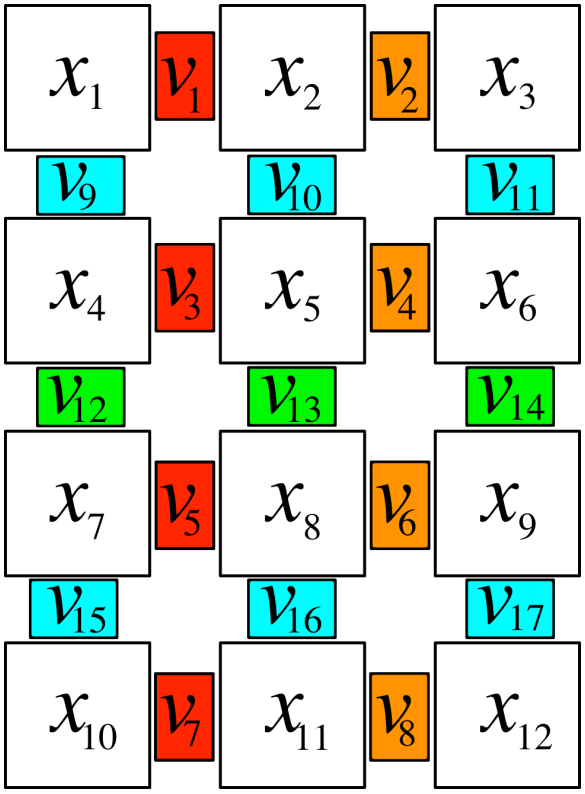


Fig. 1: Illustration of groups of elements of the dual variable $\mathbf{v}$ for a two-dimensional denoising case. Elements of $\mathbf{v}$ are updated in groups such that none of the groups affect overlapping pixels. For examples, the horizontal differences $\{v_1, v_3, v_5, v_7\}$ are one group and $\{v_2, v_4, v_6, v_8\}$ are another.

C. Denoising ($\mathbf{v}$) updates

The regularizer R penalizes the differences between each pixel and its neighbors along a predetermined set of N_r directions, *e.g.*, the three cardinal 3D directions or all thirteen 3D directions around a pixel. The finite differencing matrix $\mathbf{C} \in \mathbb{R}^{K\times N}$ computes these differences, and each of the $K = N \cdot N_\mathrm{r}$ elements of the dual variable $\mathbf{v}$ is associated with one of these differences. We update a subset $\mathbf{v}_g$ of the elements of $\mathbf{v}$:

$$\mathbf{v}_g^+ = \underset{\mathbf{v}_g}{\operatorname{argmax}} \; -\frac{1}{2\mu}\left|\left|\mathbf{v}_g - \mathbf{v}_g^-\right|\right|_{\mathbf{C}_g\mathbf{C}_g^\intercal}^2 - \mathsf{R}_g^*(\mathbf{v}_g) + \mathbf{v}_g^\intercal \mathbf{C}_g \widetilde{\mathbf{x}}. \tag{30}$$

The dual vector $\mathbf{v}$ is enormous: in our experiments, $\mathbf{v}$ is as large as thirteen images. Storing a significant fraction of $\mathbf{v}$ on the GPU is impractical, so we update only a fraction of $\mathbf{v}$ at a time. To make that update efficient, we would like the group update problem (30) to decouple into set of independent one-dimensional update problems.

1) Group design: The elements of $\mathbf{v}_g$ are coupled in (30) only by the matrix $\mathbf{C}_g\mathbf{C}_g^\intercal$. This matrix is banded and sparse: it couples differences together that involve shared pixels. This coupling is very local; Figure 1 illustrates groups that contain only uncoupled elements of $\mathbf{v}$. Updating each of these groups of differences has a "denoising" effect on almost all the pixels (up to edge conditions) in the image and involves solving a

set of independent one-dimensional subproblems.

There are many ways to form these "covering but not overlapping" groups of differences: our implementation uses the following simple "half-direction" groups. Every element of $\mathbf{v}$ corresponds to a pixel location $\mathbf{i} = (i_x, i_y, i_z)$ and an offset $\mathbf{o} = (o_x, o_y, o_z) \in \{\pm 1\}^3$. The difference that v_k represents is between the pixels located at (i_x, i_y, i_z) and $(i_x + o_x, i_y + o_y, i_z + o_z)$. The elements of $\mathbf{v}$ corresponding to a single direction all share the same offset and differ only in their pixel locations.

For each difference direction $r = 1, \ \ldots, \ N_\mathrm{r}$, we form two groups, $\mathbf{v}_{r,\mathrm{e}}$ and $\mathbf{v}_{r,\mathrm{o}}$. We assign every other difference "along the direction r" to each group. For example, if r indicates vertical differences along the y axis then we assign to $\mathbf{v}_{r,\mathrm{e}}$ differences with even i_y and those with odd i_y to $\mathbf{v}_{r,\mathrm{o}}$. In Figure 1, the cyan group $\{v_9, v_{10}, v_{11}, v_{15}, v_{16}, v_{17}\}$ and the green group $\{v_{12}, v_{13}, v_{14}\}$ partition the vertical differences in this way.

More generally, let $\mathbf{o}_r = (o_x, o_y, o_z)$ be the offset corresponding to direction r. Let $\mathbf{c}_r \in \{0, 1\}^3$ contain a single "1" in the coordinate corresponding to the first nonzero element of $\mathbf{o}_r$. For example,

$$\mathbf{o}_r = (0, 1, -1) \to \mathbf{c}_r = (0, 1, 0), \tag{31}$$

$$\mathbf{o}_r = (0, 0, 1) \to \mathbf{c}_r = (0, 0, 1). \tag{32}$$

Recall that $\mathbf{i}_k$ is the location associated with the difference v_k. We assign to $\mathbf{v}_{r,e}$ those differences v_k along the direction r such that $\mathbf{i}_k^\mathsf{T}\mathbf{c}_d$ is even.

2) One-dimensional subproblems: Having chosen $\mathbf{v}_g$ so that its elements are uncoupled by $\mathbf{C}_g\mathbf{C}_g^\mathsf{T}$, the group update problem (30) decomposes into a set of one-dimensional difference update problems for each k in the group:

$$v_k^+ = \underset{v_k}{\operatorname{argmax}} \ -\frac{1}{\mu}(v_k - \gamma)^2 - \beta_k \psi^*\left(\frac{v_k}{\beta_k}\right) \tag{33}$$

$$\gamma \triangleq v_k^- + \frac{\mu}{2}[\mathbf{C}\widetilde{\mathbf{x}}]_k, \tag{34}$$

where ψ^* is the convex conjugate of the potential function ψ. Some potential functions ψ have convenient convex conjugates that make (33) easy to directly solve:

- Absolute value: If $\psi(d) = |d|$, then ψ^* is the characteristic function of $[-1, 1]$. The solution to (33) is

$$v_k^+ = [\gamma]_{[-\beta_k, \beta_k]}. \tag{35}$$

  i.e., the projection of γ onto the closed interval $[-\beta_k, \beta_k]$.
- Huber function: If $\psi(d)$ is the Huber function,

$$\psi(d) = \begin{cases} \frac{1}{2}d^2, & |d| \le \delta \\ \delta\big(|d| - \frac{1}{2}\delta\big), & \text{else,} \end{cases} \tag{36}$$

  then its conjugate is

$$\psi^*(v) = \frac{1}{2}v^2 + \iota_{[-\delta, \delta]}(v). \tag{37}$$

  The solution to (33) is

$$v_k^+ = \left[\frac{2\beta_k\gamma}{2\beta_k + \mu}\right]_{[-\beta_k\delta, \beta_k\delta]}. \tag{38}$$

In other cases, the convex conjugate ψ^* (33) more difficult to work with analytically. For example, the Fair potential,

$$\psi(d) = \delta^2\left(\left|\frac{d}{\delta}\right| - \log\left(1 + \left|\frac{d}{\delta}\right|\right)\right), \tag{39}$$

is easier to work with in the primal domain, where it has a closed-form "shrinkage" operator, than the dual domain. To exploit potential functions with convenient shrinkage operators but inconvenient convex conjugates, we exploit the convexity of ψ^* and invoke biconjugacy:

$$\beta_k\psi^*\left(\frac{v_k}{\beta_k}\right) = \sup_{q_k} q_k v_k - \beta_k\psi(q_k). \tag{40}$$

Combining (40) and (33),

$$v_k^+ = \underset{v_k}{\operatorname{argmax}} \ \inf_{q_k} -\frac{1}{\mu}(v_k - \gamma)^2 - v_k q_k + \beta_k\psi(q_k). \tag{41}$$

By a similar Fenchel duality argument to (16), we reverse the "max" and "inf" in (41). The resulting expression involves ψ only through its "shrinkage" operator:

$$v_k^+ = \gamma - \frac{\mu}{2}q_k^+, \qquad \text{where} \tag{42}$$

$$q_k^+ = \underset{q_k}{\operatorname{argmin}} \ \frac{\mu}{4}\left(q_k - \frac{2}{\mu}\gamma\right)^2 + \beta_k\psi(q_k). \tag{43}$$

After updating a group of differences $\mathbf{v}_g$, we update the buffer $\widetilde{\mathbf{x}}$ (17):

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\mathbf{C}_g^\mathsf{T}\big(\mathbf{v}_g^+ - \mathbf{v}_g^-\big). \tag{44}$$

Because $\mathbf{v}_g$ contains variables corresponding to nonoverlapping differences, each element of $\mathbf{v}_g$ updates two pixels in $\widetilde{\mathbf{x}}$, and each pixel in $\widetilde{\mathbf{x}}$ is updated by at most one difference in $\mathbf{v}_g$.

D. Nonnegativity ($\mathbf{z}$) updates

Updating each element of the image-sized dual variable $\mathbf{z}$ helps enforce the nonnegativity constraint on the corresponding pixel of $\widetilde{\mathbf{x}}$. The dual function $\mathsf{D}^{(n)}$ is separable in the elements of $\mathbf{z}$, and the $\mathbf{z}$ update is

$$\mathbf{z}^+ = \underset{\mathbf{z}}{\operatorname{argmax}} \ -\frac{1}{2\mu}\big|\big|\mathbf{z} - \mathbf{z}^-\big|\big|^2 + \mathbf{z}^\mathsf{T}\widetilde{\mathbf{x}} - \mathsf{N}^*(\mathbf{z}) \tag{45}$$

$$= \sum_k -\frac{1}{2\mu}(z_k - \eta_k)^2 - \iota_k^*(z_k), \qquad \text{where} \tag{46}$$

$$\eta_k \triangleq z_k^- + \mu[\widetilde{\mathbf{x}}]_k. \tag{47}$$

The dual characteristic function ι_k^* is also a characteristic function, but on the nonpositive numbers:

$$\iota_k^*(z_k) = \begin{cases} \infty, & z_k > 0 \\ 0, & \text{else.} \end{cases} \tag{48}$$

We solve (46) by clamping η_k to the nonpositive numbers:

$$z_k^+ = [\eta_k]_{(-\infty, 0]}. \tag{49}$$

After updating $\mathbf{z}$ (46) we update the buffer $\widetilde{\mathbf{x}}$:

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\big(\mathbf{z}^+ - \mathbf{z}^-\big). \tag{50}$$

E. Warm starting

The dual variable updates in Sections II-B, II-C and II-D find values for the dual variables, $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ that approximately maximize the dual update problem (19). We use these dual variables and the induced solution $\widetilde{\mathbf{x}}^{(n+1)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right)$, stored in the buffer $\widetilde{\mathbf{x}}$, to determine $\mathbf{x}^{(n+1)}$:

$$\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right) = \widetilde{\mathbf{x}}, \qquad (51)$$

then re-form the outer update problem (5) and repeat.

We initialize our algorithm with all the dual variables set to zero. We could also reset the dual variables to zero every outer iteration, but this empirically led to slow convergence. Instead, mirroring a practice in other alternating directions algorithms, we warm-start each outer iteration with the previous iteration's dual variable values. This has an extrapolation-like effect on the buffer $\widetilde{\mathbf{x}}$:

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^{(n+2)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right) \qquad (52)$$

$$= \mathbf{x}^{(n+1)} - \frac{1}{\mu}\left(\mathbf{A}^\intercal \mathbf{u}^{(n+1)} + \mathbf{C}^\intercal \mathbf{v}^{(n+1)} + \mathbf{z}^{(n+1)}\right) \qquad (53)$$

$$= \mathbf{x}^{(n)} - \frac{2}{\mu}\left(\mathbf{A}^\intercal \mathbf{u}^{(n+1)} + \mathbf{C}^\intercal \mathbf{v}^{(n+1)} + \mathbf{z}^{(n+1)}\right) \qquad (54)$$

$$= \widetilde{\mathbf{x}}^{-} + \left(\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}\right). \qquad (55)$$

After initializing $\widetilde{\mathbf{x}}$ with this "extrapolated" value, subsequent iterated dual updates refine the update. This extrapolation is just an initial condition for the iterative algorithm solving the dual problem (19). If the dual function $\mathsf{D}^{(n+1)}$ were maximized exactly then this extrapolation would have no effect on $\mathbf{x}^{(n+2)}$.

This section outlined the mathematical framework of our proposed CT reconstruction algorithm. Using duality and group coordinate ascent, we decomposed the process of solving the original reconstruction problem (1) into an iterated series of optimization steps, each considering only a portion of the original cost function. The next section describes how we implemented these operations on the GPU.

III. Implementation

For implementing the algorithm described in Section II, GPUs have two important properties:

- GPUs can provide impressive speedups for highly parallel workloads, and;
- GPUs often have much less memory than their host computers.

The first property means that algorithm designers should favor independent operations with regular memory accesses. Our proposed algorithm consists of five operations, each of which has an efficient GPU implementation:

- Tomography update (27): Updating the tomography dual variables corresponding to a group of projection views, $\mathbf{u}_g$, consists of projecting those views, a few vector operations, and then backprojecting those views. Our algorithm relies on projections and backprojections of subsets of the data, and it should be usable with any system model that is suitable for OS methods. Implementing an efficient CT system model on the GPU is nontrivial, and we rely on previous work [25]–[27]. In our experiments, we use the separable footprints CT system model [28]. Our implementation uses thousands of threads for both projection and backprojection to exploit the GPU's parallelism: we use one thread per detector cell in projection and one thread per voxel in backprojection.
- Denoising update (30): Updating a "half-difference" of elements of $\mathbf{v}$ is also highly parallel. We assign one thread to each element dual variable being updated; each thread updates two neighboring pixels of the image $\widetilde{\mathbf{x}}$. The workload for each thread is independent, and memory accesses are both local and regular.
- The nonnegativity update (49) and warm starting operation (55) both consist entirely of separable, parallelizable vector operations.

The GPU's memory constraints are very relevant for large imaging problems. We performed the experiments in Section IV on a machine with four NVIDIA Tesla C2050s having 3 GB of memory apiece. The wide-cone axial experiment in Section IV-D requires about 894 MB each for the noisy data $\mathbf{y}$ and the statistical weights $\mathbf{W}$ when stored in single-precision 32-bit floating point. Storing the regularizer parameters $\{\beta_k\}$ and a single image $\mathbf{x}$ would take an additional 907 MB apiece. Altogether, storing one image and the parameters of the reconstruction problem would take about 2.7 GB, leaving no room for the algorithm to store any additional state on a single GPU!

Because the X-ray CT reconstruction problem (1) is so memory-intensive, many algorithms will need to periodically transfer some data between the GPU and the host computer. If performed naïvely, these transfers can have a significant effect on algorithm speed. Fortunately, modern GPUs can perform calculations and memory transfers simultaneously, so we can "hide" the latency of these transfers to some degree.

A. Streaming

Our algorithm has many variables: the dual variable $\mathbf{v}$ alone is often as large as 13 image volumes. This is far too much to fit simultaneously on the GPU for many realistic problem sizes. Fortunately, each of proposed algorithm's operations requires a comparatively small subset of the data. For example, performing a tomography update requires only $\widetilde{\mathbf{x}}$, and the data, weights and dual variables corresponding to the view being updated.

The algorithm in Figure 2 allocates on the GPU only

- a buffer containing $\widetilde{\mathbf{x}}$,
- an image-sized buffer for storing $\mathbf{z}$ or a subset of $\mathbf{v}$,
- the regularizer parameters $\{\beta_k\}$,

and several negligibly small view-sized buffers on the GPU. The dual variables are stored primarily on the host computer and transferred to and from the GPU as needed.

The tomography update requires a relatively small amount of data: several view-sized buffers. Even for the wide-cone reconstruction in Section IV-D, each tomography update requires less than 4 MB of projection-domain data. The projection

Initialize CPU memory: $\mathbf{x}^{(0)} = \mathbf{x}_{\text{FBP}}$, $\mathbf{u} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$, $\mathbf{z} = \mathbf{0}$,
Initialize GPU memory: $\underbrace{\widetilde{\mathbf{x}}_{\text{GPU}} = \mathbf{x}^{(0)},\ \mathbf{b}_{\text{GPU}} = \mathbf{0}}_{\text{Image sized}}$, $\underbrace{\mathbf{u}_{g,\text{ GPU}} = \mathbf{0},\ \mathbf{y}_{g,\text{ GPU}} = \mathbf{0},\ \mathbf{m}_{g,\text{ GPU}} = \mathbf{0},\ \mathbf{w}_{g,\text{ GPU}} = \mathbf{0}}_{\text{Projection view sized}}$.

Repeat iterations n, until convergence:	Transfer	Computation on GPU
Send nonnegativity variables to GPU	$\mathbf{z}^- \rightarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (27)
Nonnegativity update		Update $\mathbf{z}$ (49)
Send nonnegativity variables to host	$\mathbf{z}^+ \leftarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (27)
Repeat N_{denoise} times:		
Choose half-difference $\mathbf{v}_g$ randomly		
Send $\mathbf{v}_g$ to GPU	$\mathbf{v}_g^- \rightarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (27)
Denoising update		Denoising update (30)
Send updated $\mathbf{v}_g$ to host	$\mathbf{v}_g^+ \leftarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (27)
Send $\mathbf{x}^{(n)}$ to GPU	$\mathbf{x}^{(n)} \rightarrow \mathbf{b}_{\text{GPU}}$	
Warm-start $\widetilde{\mathbf{x}}^{(n+1)}$		$\mathbf{b}_{\text{GPU}} = \widetilde{\mathbf{x}}_{\text{GPU}}$; warm-start $\widetilde{\mathbf{x}}_{\text{GPU}}$ (55)
Send $\mathbf{x}^{(n+1)}$ to host	$\mathbf{x}^{(n+1)} \leftarrow \mathbf{b}_{\text{GPU}}$	

Fig. 2: Pseudocode for the proposed algorithm. The buffer $\widetilde{\mathbf{x}}_{\text{GPU}}$ is updated on the GPU using (17) in every step as the dual variables are updated, and the buffer $\mathbf{b}_{\text{GPU}}$ stores other variables as they are needed for computation on the GPU. Updating each view of $\mathbf{u}$ involves a one-view projection and backprojection and transferring a small amount of memory. The view weights $\mathbf{w}_g$, data $\mathbf{y}_g$, dual variables $\mathbf{u}_g$, and majorizer weights $\mathbf{m}_g$ are transferred to the GPU prior to updating $\mathbf{u}_g$. Only the updated $\mathbf{u}_g$ needs to be transferred back to the host afterwards.

and backprojection involved in the tomography update take much longer to perform than it takes to transfer the dual variables and weights to and from the GPU. Therefore, the tomography update is computation-bound. On the other hand, the nonnegativity, denoising, and warm-start operations require whole images to be transferred to and from the GPU with relatively small amounts of computation. The speed of these operations is bounded by the latency of data transfers between the host computer and the GPU.

Modern GPUs can perform computations and transfer memory concurrently. This allows us to "hide" some of the cost of latency-bound operations by performing computation-bound operations and vice versa. The pseudocode in Figure 2 interleaves computation-bound and transfer-bound operations. After each large memory transfer is begun, the algorithm performs N_{tomo} tomography updates. These tomography updates serve to "hide" the latency of the large memory transfer by performing useful work instead of waiting for the relatively slow memory transfer to finish. Section III-C discusses selecting N_{tomo} and other algorithm parameters.

B. Multiple-device implementation

Besides providing more computational resources, implementing a CT reconstruction algorithm on multiple GPUs can reduce the memory burden on each device. Many "distributed" algorithms either store additional variables on each node and/or perform redundant calculations to avoid very expensive inter-node communications [29]–[32]. These designs assume that communication between devices is extremely expensive. It may be tempting to view multiple-GPU programming as a "distributed" setting, but at least in CT reconstruction, frequent communication between the host computer and the GPU(s) seems necessary due to GPU memory constraints. Adding additional GPUs that regularly communicate to the host need not significantly increase the total amount of communication the algorithm performs. Instead of using a more sophisticated "distributed" algorithm framework [30], we distribute the memory and computation of the single-GPU algorithm over multiple devices in a straightforward way.

Let x and y be the transaxial axes of the image volume and z be the axial direction; *i.e.*, z is parallel to the CT scanner's axis of rotation. Similar to [33, Appendix E], we divide all image-sized buffers into N_{device} chunks transaxially, *e.g.*, along the y direction. This approach differs from [30], [31], where the image is divided axially along z. Each device also stores two $N_x N_z$-pixel padding buffers. Because the image-sized buffers are the largest buffers our proposed algorithm stores on the GPU, this decomposition effectively reduces the memory burden on each device by a factor of almost N_{device}.

1) Tomography update: The buffer $\widetilde{\mathbf{x}}$ is distributed across multiple GPUs. Fortunately, the tomography update (27) is linear in $\widetilde{\mathbf{x}}$. When updating the group of dual variables $\mathbf{u}_g$, each device projects its chunk of $\widetilde{\mathbf{x}}$ and sends the projection to the host computer. The host accumulates these projections, performs the update (27), and transmits the dual update $\mathbf{u}_g^+ - \mathbf{u}_g^-$ back to each device. Each device backprojects the update into its chunk of the volume, updating the distributed $\widetilde{\mathbf{x}}$.

2) Denoising update: Every element of the dual variable $\mathbf{v}$ couples two pixels together. Most of these pairs of pixels lie on only one device; in these cases, the denoising update is separable and requires no additional communication between the GPUs. However, some of the elements of $\mathbf{v}$ couple pixels that are stored on different GPUs. Prior to performing the update for these elements, the algorithm must communicate the pixels on the GPU boundaries between devices.

Fortunately, such communication is needed for only roughly a quarter of the denoising updates. Most of the "half-difference" groups in which $\mathbf{v}$ is updated require no

communication. For example, in Figure 1 suppose that $\{x_1, \ \ldots, \ x_6\}$ were stored on one device and $\{x_7, \ \ldots, \ x_{12}\}$ are stored on another. Updating the green group of differences $\{v_{12}, v_{13}, v_{14}\}$ would require communication between the devices, but updating the cyan group $\{v_9, v_{10}, v_{11}, v_{15}, v_{16}, v_{17}\}$ would not.

3) Nonnegativity update and warm-starting: The nonnegativity update and warm-stating operation are both separable in the elements of the dual variables and $\widetilde{\mathbf{x}}$, so implementing these operations on multiple devices is straightforward.

C. Parameter selection

There are three parameters in the algorithm listed in Figure 2: N_{denoise}, N_{tomo} and μ. This section gives the heuristics we used to set the values of these parameters.

Similar to how OS algorithms compute an approximate gradient using only a subset of the projection views, the proposed algorithm performs an outer update (*i.e.*, increments the iteration n) after updating about $N_{\text{view}}/N_{\text{subset}}$ views of $\mathbf{u}$, chosen randomly with replacement. In an outer iteration, the algorithm also performs N_{denoise} half-difference denoising updates. We heuristically set N_{denoise} to be large enough that the expected number of outer iterations between updating an element of $\mathbf{v}$ is about one. Because each denoising update modifies half of the elements of $\mathbf{v}$ corresponding to a single direction, this means $N_{\text{denoise}} \approx 2N_{\text{r}}$, where N_{r} is the number of directions along which the regularizer penalizes differences. For the common case of penalizing differences in 13 directions around each pixel (as in our experiments), we set $N_{\text{denoise}} \approx 26$.

This yields the following relationship:

$$N_{\text{tomo}} = \frac{N_{\text{view}}}{2N_{\text{denoise}}N_{\text{subset}}}. \tag{56}$$

In the shoulder case below, $N_{\text{view}} = 6852$. We set $N_{\text{subset}} = 18$ and $N_{\text{denoise}} = 23$, yielding $N_{\text{tomo}} = 8$. The wide cone case had $N_{\text{view}} = 984$; we used $N_{\text{denoise}} = 27$ and $N_{\text{subset}} = 6$, thus, $N_{\text{tomo}} = 3$. A more principled method to select these parameters is future work.

We chose μ using the mean of the entries of the diagonal matrix $\mathbf{MW}$, where $\mathbf{W}$ contains the weights in the data-fit term and $\mathbf{M}$ contains the entries of all the diagonal majorizers for the tomography update (28):

$$\mu = \frac{\sum_{i=1}^{M} [\mathbf{MW}]_{ii}}{4M}. \tag{57}$$

This heuristic is intended to yield a well-conditioned tomography update (27). Smaller μ would make the outer proximal majorization tighter (5) at the cost of making the dual problem (19) possibly more ill-conditioned.

IV. Experiments

This section compares the proposed algorithm to several state of the art accelerated versions of the popular OS algorithm [12]–[14]. All algorithms were implemented with the OpenCL GPU programming interface and the Rust programming language. Experiments were run on a machine with 48 GB of RAM and four aging NVIDIA Tesla C2050s with 3 GB of memory apiece. To measure how well the algorithms performed on multiple devices we ran each algorithm using 1, 2 and 4 GPUs. Preliminary experiments on an NVIDIA Kepler K5200 (see the supplementary material) indicate that all the algorithms run faster on newer hardware, but their relative speeds are unchanged.

A. Ordered subsets

OS algorithms are first-order methods that approximate the computationally expensive gradient of the data-fit term $\nabla \mathsf{L}(\mathbf{Ax})$ using a subset of the data [12]–[14]. Without relaxation, this approximation can lead to divergence, but in our experiments we chose parameters that empirically led to limit cycles near the solution.

Our implementation of the OS algorithms stored the following variables on each GPU:

- the current image $\mathbf{x}$,
- the coefficients of the diagonal majorizer $\mathbf{D} \succeq \mathbf{A}^\intercal\mathbf{WA}$:

$$\mathbf{D} = \operatorname*{diag}_j \Big\{ \left[\mathbf{A}^\intercal\mathbf{WA}\mathbf{1}\right]_j \Big\}, \tag{58}$$

- an accumulator for the current search direction,
- the regularizer parameters $\{\beta_k\}$, and
- an additional image-sized variable to store the momentum state, if applicable [13].

The OS methods require more image-sized buffers than our proposed algorithm. We divided these image-sized volumes across multiple GPUs transaxially, so the memory burden for each device decreases almost linearly with the number of devices. The devices must communicate pixels that lie on an edge between devices before computing a regularizer gradient; this happens N_{subset} times per iteration.

The OS methods also must compute the majorizer $\mathbf{D}$ in (58) from the patient-dependent statistical weights $\mathbf{W}$ before beginning to iterate. This requires a CT projection and a backprojection that takes about as much time as an iteration. We do not count this time in our experiments. On the other hand, the majorizers used by the proposed algorithm are nominally independent of patient data and depend only on the scanner geometry through $\mathbf{A}_g\mathbf{A}_g^\intercal$. Because the majorizers $\{\mathbf{M}_g\}$ in (28) can be precomputed before the scan, but the OS algorithms must compute $\mathbf{D}$ (58) after the scan but before beginning to iterate, the proposed algorithm could be considered be an additional iteration faster than the OS-based methods.

B. Figures of merit

We ran experiments using two datasets: a helical shoulder scan using real patient data (Section IV-C) and a wide-cone axial scan using simulated data in (Section IV-D). For both cases we measured the progress of all algorithms tested towards a converged reference $\widehat{\mathbf{x}}$ using the root mean squared difference (RMSD) over an N_{ROI}-pixel region of interest (ROI):

$$\text{RMSD}\Big(\mathbf{x}^{(n)}\Big) = \sqrt{\frac{\left|\left|\mathbf{x}^{(n)} - \widehat{\mathbf{x}}\right|\right|_{\mathbf{M}_{\text{ROI}}}^2}{N_{\text{ROI}}}}. \tag{59}$$

The diagonal binary masking matrix $\mathbf{M}_{\text{ROI}}$ discards "end slices" that are needed due to the "long object problem" in cone-beam CT reconstruction but not used clinically [34].

We compared the proposed algorithm and the OS algorithms using wall-clock time and "equivalent iterations," or equits [7]. Because the most computationally intensive part of these algorithms is projection and backprojection, one equit corresponds to computing roughly the same number of forward and backprojections:

- For OS, one equit corresponds to a loop through all the data.
- For the proposed algorithm, one equit corresponds to N_{subset} iterations; *e.g.*, $\mathbf{x}^{(N_{\text{subset}})}$ and $\mathbf{x}^{(2N_{\text{subset}})}$ are the outputs of successive equits. Over this time, our algorithm computes about N_{view} forward and backprojections.

The proposed algorithm performs more denoising updates than OS and transfers more memory between the host and GPU in an equit, so each equit of the proposed algorithm takes longer to perform. Nonetheless, as the following experiments show, the proposed algorithm converges considerably more quickly than the OS algorithms in terms of runtime.

C. Helical shoulder scan

Our first reconstruction experiment uses data from a helical scan provided by GE Healthcare. The data were 6852 views with 888 channels and 32 rows. We reconstructed the image on a $512 \times 512 \times 109$-pixel grid. We used a weighted squared ℓ_2-norm data-fit term with weights proportional to estimates of the inverses of the measurements' variances [1]. The regularizer penalized differences along all 13 directions (*i.e.*, 26 3D neighbors) with the Fair potential function (39) with $\delta = 10$ Hounsfield units (HU), and the $\{\beta_k\}$ were provided by GE Healthcare. All iterative algorithms were initialized using the filtered backprojection image in Figure 3a. Figure 3b shows an essentially converged reference, generated by running thousands of iterations of momentum-accelerated separable quadratic surrogates [13] (*i.e.*, OS with one subset).

We ran the proposed algorithm with $N_{\text{denoise}} = 23$ and $N_{\text{tomo}} = 8$. We also ran ordered subsets with 12 subsets (OS) [12], OS with Nesterov's momentum [13] (FGM), and OS with a faster acceleration [14] (OGM) on one, two and four GPUs. Figure 4 shows RMSD in Hounsfield units against time and equivalent iteration.

Figure 4a shows the proposed algorithm converging considerably faster than the OS-type algorithms in terms of equits, and unlike the OS-type algorithms will converge to a solution $\widehat{\mathbf{x}}$ if the conditions in Appendix C are satisfied. Figure 4c shows that the proposed algorithm on one GPUs achieves early-iteration speed comparable to the fastest OS algorithm with four GPUs.

Table I lists several timings for the algorithms in this experiment. Although the OS algorithms achieved more dramatic speedups using multiple devices than the proposed algorithm, additional devices did help accelerate convergence. Figures 3c and 3d show images from both algorithms on four devices after about five minutes of computation. The proposed algorithm produced an image that much more closely matches the converged reference.

Algorithm	Per equit	Time to converge within 5 HU RMSD	Time to converge within 2 HU RMSD
OGM-1	114 sec.	21.5 min.	58.6 min.
OGM-2	62 sec.	11.7 min.	29.8 min.
OGM-4	38 sec.	7.2 min.	18.2 min.
Proposed-1	130 sec.	9.4 min.	16.2 min.
Proposed-2	82 sec.	5.5 min.	9.8 min.
Proposed-4	57 sec.	3.8 min.	6.8 min.

TABLE I: Timings for the helical case in Section IV-C.

Algorithm	Per equit	Time to converge within 5 HU RMSD	Time to converge within 2 HU RMSD
OGM-2	85 sec.	16.3 min.	–
OGM-4	50 sec.	9.5 min.	19.9 min.
Proposed-2	126 sec.	8.6 min.	15.6 min.
Proposed-4	98 sec.	6.3 min.	11.0 min.

TABLE II: Timings for the axial case in Section IV-D.

D. Wide-cone axial simulation

Our second experiment is a wide-cone axial reconstruction with a simulated phantom. We simulated a noisy scan of the XCAT phantom [35] taken by a scanner with 888 channels and 256 rows over a single rotation of 984 views. Images were reconstructed onto a $718 \times 718 \times 440$-pixel grid. As in Section IV-C, we used a quadratic data-fit term, the regularizer used the Fair potential and penalized differences along all 13 neighboring directions, and the regularizer weights $\{\beta_k\}$ were computed using [16]. All iterative algorithms were initialized with the filtered backprojection image in Figure 5a. An essentially converged reference image is shown in Figure 5b.

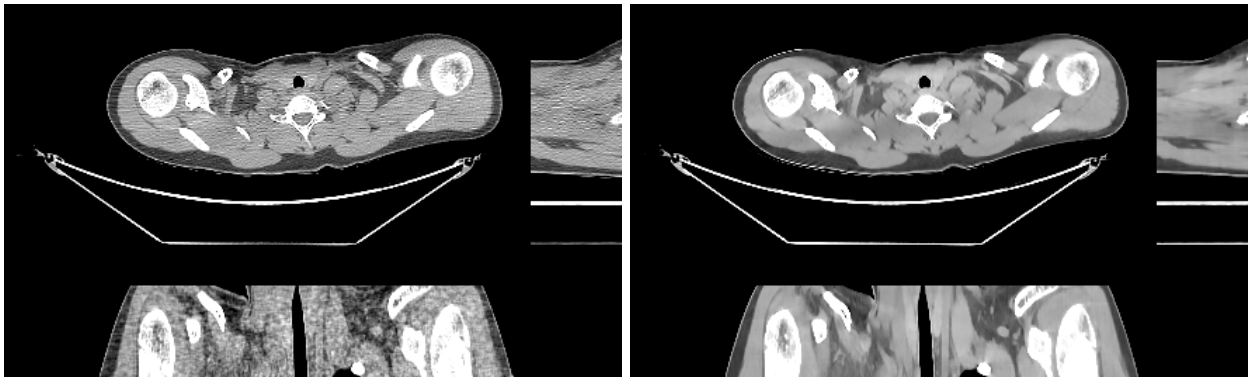
This problem was too large to fit on one 3 GB GPU, so we present results for two and four GPUs. We ran the same set of OS algorithms as the previous experiment with 12 subsets. The proposed algorithm used $N_{\text{denoise}} = 27$ and $N_{\text{tomo}} = 3$.

Figures 6a and 6c show the progress of the tested algorithms towards the converged reference. The proposed algorithm running on two devices is about as fast as OS-OGM running on four devices, and additional devices accelerate convergence even more. Figures 5c and 5d show outputs from OS-OGM and the proposed algorithm after about five minutes. After five minutes, the OS algorithm still contains noticeable streaks that the dual algorithm has already removed. At this point, both algorithms have significant distance to the reference at the end slices of the image.

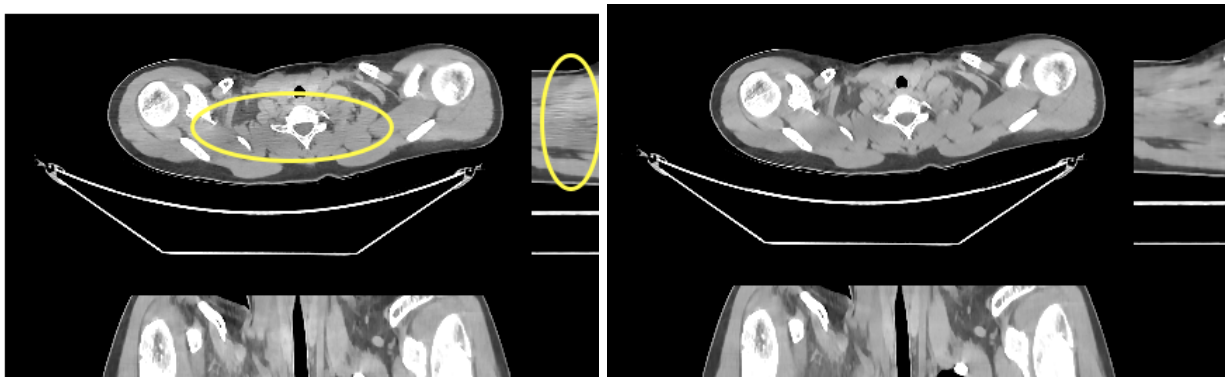
Table II lists timings for OS-OGM and the proposed algorithm. The trends are similar to the smaller helical case in Table I. The OS algorithms scale better ($1.7\times$ faster) than the proposed algorithm ($1.2\times$ faster) from two to four GPUs, but the acceleration provided by the proposed algorithm is enough to compensate for lower multi-device parallelizability.

V. Conclusions and future work

We presented a novel CT reconstruction algorithm that uses alternating updates in the dual domain. The proposed algorithm is fast in terms of per-iteration speed and "wall clock" runtime, and it converges more quickly than state of

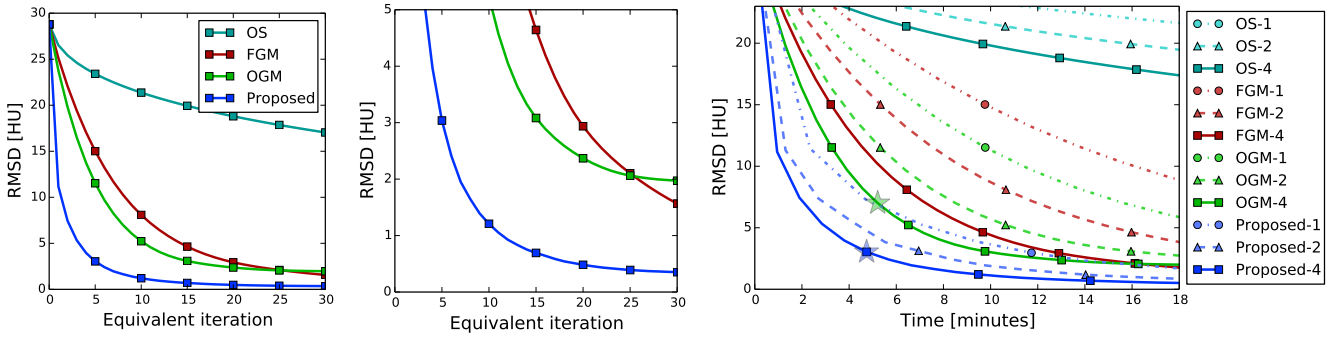


(a) Filtered backprojection (b) Reference



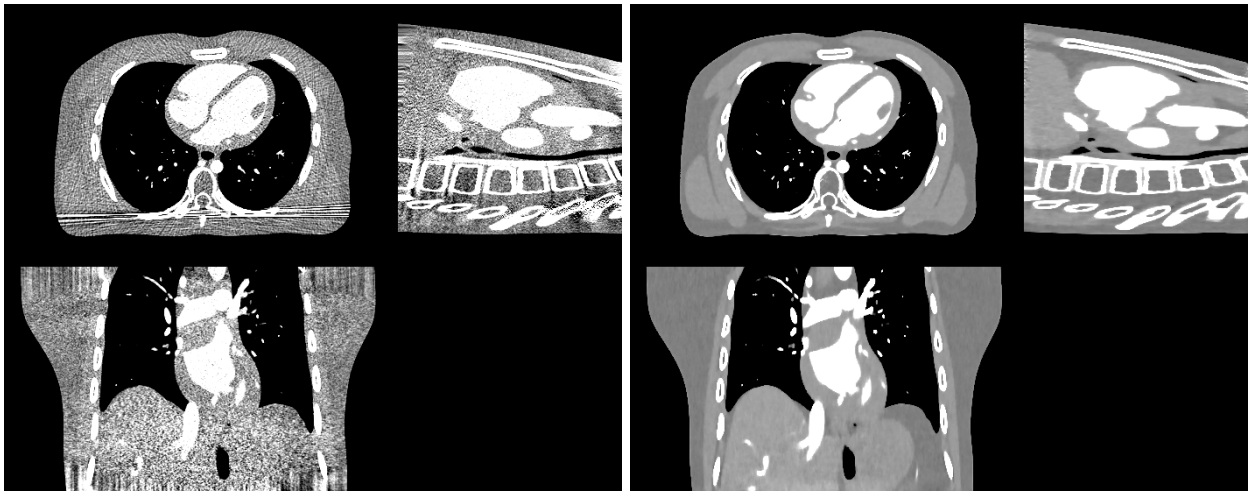
(c) OS-OGM with 4 GPUs after 8 equits (5.2 minutes) (d) Proposed with 4 GPUs after 5 equits (4.8 minutes)

Fig. 3: Top row: initial FBP and reference images for the helical shoulder case in Section IV-C. Bottom row: images from the proposed algorithm and the state of the art OS-OGM algorithm on 4 GPUs after about 5 minutes; yellow ovals indicate regions with significant differences. Images were trimmed and displayed in a $[800,\ 1200]$ modified Hounsfield unit window. Each panel shows transaxial, coronal, and sagittal slices through the 3D volume.

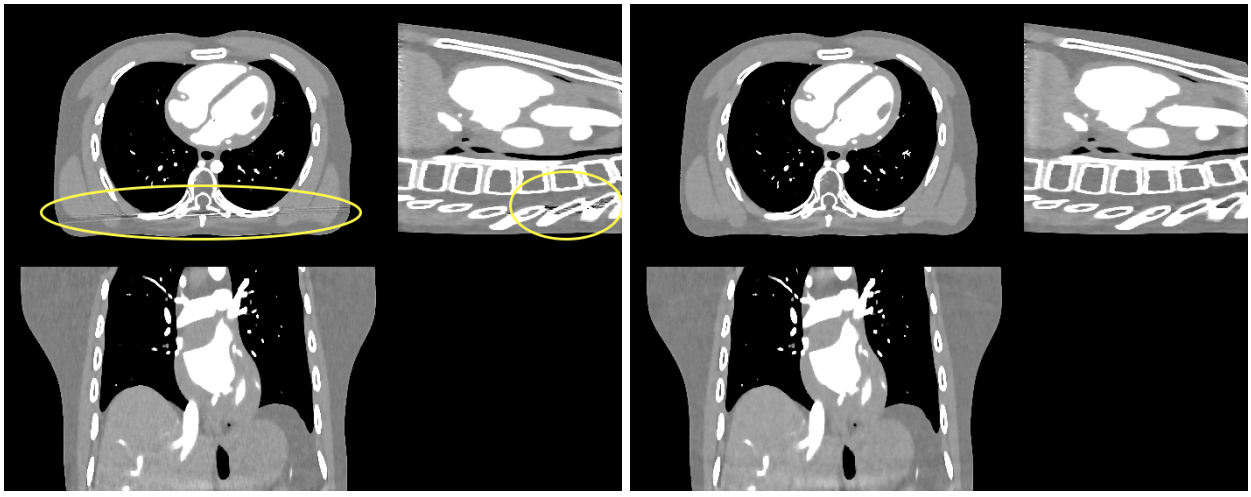


(a) RMSD *vs.* equit (b) RMSD *vs.* equit (zoomed) (c) RMSD *vs.* time

Fig. 4: Convergence plots for the helical shoulder case in Section IV-C. Markers are placed every five equits. The stars in Figure 4c correspond to the images shown in Figures 3c and 3d. The proposed algorithm on one device converges about as quickly as the state of the art OS-OGM algorithm does on four devices. Additional devices provide further acceleration.

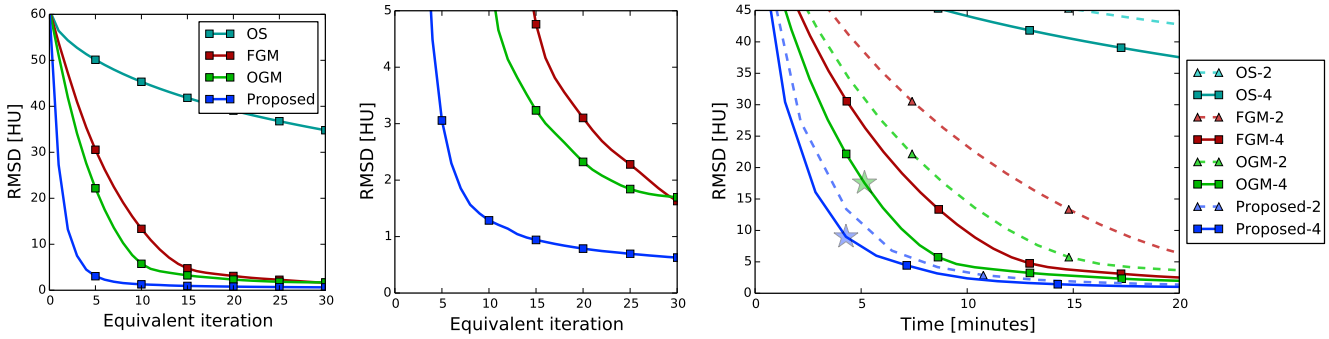


(a) Filtered backprojection (b) Reference



(c) OS-OGM with 4 GPUs after 6 equits (5.2 minutes) (d) Proposed with 4 GPUs after 3 equits (4.7 minutes)

Fig. 5: Top row: initial FBP and reference images for the wide-cone axial case in Section IV-D. Bottom row: images from the proposed algorithm and the state of the art OS-OGM algorithm on 4 GPUs after about 5 minutes; yellow ovals indicate regions with significant differences. Images were trimmed and are displayed in a $[800,\ 1200]$ modified Hounsfield unit window. Each panel shows transaxial, coronal, and sagittal slices through the 3D volume.



(a) RMSD *vs.* equit (b) RMSD *vs.* equit (zoomed) (c) RMSD *vs.* time

Fig. 6: Convergence plots for the wide-cone axial case in Section IV-D. Markers are placed every five equits. The stars in Figure 6c correspond to the images shown in Figures 5c and 5d. The proposed algorithm converges about as quickly on two devices as OS-OGM does on four. Additional devices accelerate convergence.

the art OS algorithms. If the inner updates are performed with sufficient accuracy the algorithm converges to the true solution of the statistical reconstruction problem, and it can handle a wide range of regularizers including the nondifferentiable total variation regularizer.

The algorithm also maps well onto the GPU. Many of its steps are highly parallelizable and perform regular memory accesses. Although the algorithm stores many variables in the host computer's memory, the amount of memory required for each update is relatively small, and we hide the latency of transferring variables to and from the GPU by performing computation-bounded operations. Finally, the proposed algorithm is easily adapted for multiple GPUs, providing further acceleration and decreasing the memory burden on each GPU.

Due to communication overhead, the acceleration provided by adding additional GPUs showed diminishing returns. To achieve further acceleration, multiple computers (or groups of GPUs on a single node) may need to be combined using a "distributed" algorithm framework [29], [30]. How to best adapt the proposed algorithm to these frameworks is future work.

The proposed algorithm introduces a dual variable for each difference penalized by the edge-preserving regularizer R. While this memory cost is not too great for a modern computer when regularizing the 13 neighbors around each pixel, increasing the number of differences computed may make the proposed approach infeasible. Consequently, adapting the proposed algorithm for patch-based or nonlocal regularizers may be challenging.

The random process we use for choosing which groups of the tomography dual variable $\mathbf{u}$ and denoising dual variable $\mathbf{v}$ to update is basic and almost certainly suboptimal. A more sophisticated strategy may provide additional acceleration. Different majorizers $\mathbf{M}_g$ for the tomography update (27) and more sophisticated methods to select the algorithm parameters N_{tomo} and N_{denoise} are other interesting areas for future work.

REFERENCES

[1] J.-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh, "A three-dimensional statistical approach to improved image quality for multi-slice helical CT," *Med. Phys.*, vol. 34, no. 11, pp. 4526–44, Nov. 2007.

[2] J. A. Fessler, "Statistical image reconstruction methods for transmission tomography," in *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis*, M. Sonka and J. M. Fitzpatrick, Eds. Bellingham: SPIE, 2000, pp. 1–70.

[3] I. A. Elbakri and J. A. Fessler, "Statistical image reconstruction for polyenergetic X-ray computed tomography," *IEEE Trans. Med. Imag.*, vol. 21, no. 2, pp. 89–99, Feb. 2002.

[4] L. Fu, Z. Yu, J.-B. Thibault, B. D. Man, M. G. McGaffin, and J. A. Fessler, "Space-variant channelized preconditioner design for 3D iterative CT reconstruction," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2013, pp. 205–8. [Online]. Available: proc/13/web/fu-13-svc.pdf

[5] N. H. Clinthorne, T. S. Pan, P. C. Chiao, W. L. Rogers, and J. A. Stamos, "Preconditioning methods for improved convergence rates in iterative reconstructions," *IEEE Trans. Med. Imag.*, vol. 12, no. 1, pp. 78–83, Mar. 1993.

[6] J. A. Fessler, E. P. Ficaro, N. H. Clinthorne, and K. Lange, "Grouped-coordinate ascent algorithms for penalized-likelihood transmission image reconstruction," *IEEE Trans. Med. Imag.*, vol. 16, no. 2, pp. 166–75, Apr. 1997.

[7] Z. Yu, J.-B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh, "Fast model-based X-ray CT reconstruction using spatially non-homogeneous ICD optimization," *IEEE Trans. Im. Proc.*, vol. 20, no. 1, pp. 161–75, Jan. 2011.

[8] S. Ramani and J. A. Fessler, "A splitting-based iterative algorithm for accelerated statistical X-ray CT reconstruction," *IEEE Trans. Med. Imag.*, vol. 31, no. 3, pp. 677–88, Mar. 2012.

[9] H. Nien and J. A. Fessler, "Accelerating ordered-subsets X-ray CT image reconstruction using the linearized augmented Lagrangian framework," in *Proc. SPIE 9033 Medical Imaging 2014: Phys. Med. Im.*, 2014, p. 903332.

[10] ——, "Fast X-ray CT image reconstruction using a linearized augmented Lagrangian method with ordered subsets," *IEEE Trans. Med. Imag.*, vol. 34, no. 2, pp. 388–99, Feb. 2015.

[11] L. Pfister and Y. Bresler, "Adaptive sparsifying transforms for iterative tomographic reconstruction," in *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, 2014, pp. 107–10.

[12] H. Erdoğan and J. A. Fessler, "Ordered subsets algorithms for transmission tomography," *Phys. Med. Biol.*, vol. 44, no. 11, pp. 2835–51, Nov. 1999.

[13] D. Kim, S. Ramani, and J. A. Fessler, "Combining ordered subsets and momentum for accelerated X-ray CT image reconstruction," *IEEE Trans. Med. Imag.*, vol. 34, no. 1, pp. 167–78, Jan. 2015.

[14] D. Kim and J. A. Fessler, "Optimized momentum steps for accelerating X-ray CT ordered subsets image reconstruction," in *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, 2014, pp. 103–6.

[15] J. W. Stayman and J. A. Fessler, "Regularization for uniform spatial resolution properties in penalized-likelihood image reconstruction," *IEEE Trans. Med. Imag.*, vol. 19, no. 6, pp. 601–15, June 2000.

[16] J. H. Cho and J. A. Fessler, "Regularization designs for uniform spatial resolution and noise properties in statistical image reconstruction for 3D X-ray CT," *IEEE Trans. Med. Imag.*, vol. 34, no. 2, pp. 678–89, Feb. 2015.

[17] M. McGaffin and J. A. Fessler, "Edge-preserving image denoising via group coordinate descent on the GPU," *IEEE Trans. Im. Proc.*, vol. 24, no. 4, pp. 1273–81, Apr. 2015.

[18] M. V. Afonso, Jośe. M. Bioucas-Dias, and Mário. A. T. Figueiredo, "Fast image recovery using variable splitting and constrained optimization," *IEEE Trans. Im. Proc.*, vol. 19, no. 9, pp. 2345–56, Sept. 2010.

[19] M. G. McGaffin and J. A. Fessler, "Duality-based projection-domain tomography solver for splitting-based X-ray CT reconstruction," in *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, 2014, pp. 359–62.

[20] ——, "Fast GPU-driven model-based X-ray CT image reconstruction via alternating dual updates," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2015, pp. 312–5. [Online]. Available: proc/15/web/mcgaffin-15-fgd.pdf

[21] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1-3, pp. 293–318, Apr. 1992.

[22] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *J. Mach. Learning Res.*, vol. 14, pp. 567–99, Feb. 2013. [Online]. Available: http://jmlr.org/papers/v14/shalev-shwartz13a.html

[23] K. Lange, D. R. Hunter, and I. Yang, "Optimization transfer using surrogate objective functions," *J. Computational and Graphical Stat.*, vol. 9, no. 1, pp. 1–20, Mar. 2000. [Online]. Available: http://www.jstor.org/stable/info/1390605?seq=1

[24] M. W. Jacobson and J. A. Fessler, "An expanded theoretical treatment of iteration-dependent majorize-minimize algorithms," *IEEE Trans. Im. Proc.*, vol. 16, no. 10, pp. 2411–22, Oct. 2007.

[25] M. Wu and J. A. Fessler, "GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2011, pp. 56–9. [Online]. Available: http://www.fully3d.org

[26] A. S. Wang, J. W. Stayman, . Y. Otake, G. Kleinszig, S. Vogt, and J. H. Siewerdsen, "Nesterov's method for accelerated penalized-likelihood statistical reconstruction for C-arm cone-beam CT," in *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, 2014, pp. 409–13.

[27] D. Matenine, S. Hissoiny, and P. Després, "GPU-accelerated few-view CT reconstruction using the OSC and TV techniques," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2011, pp. 76–9.

[28] Y. Long, J. A. Fessler, and J. M. Balter, "3D forward and back-projection for X-ray CT using separable footprints," *IEEE Trans. Med. Imag.*, vol. 29, no. 11, pp. 1839–50, Nov. 2010.

[29] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. & Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.

[30] D. Kim and J. A. Fessler, "Distributed block-separable ordered subsets for helical X-ray CT image reconstruction," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2015, pp. 138–41. [Online]. Available: proc/15/web/kim-15-dbs.pdf

[31] J. M. Rosen, J. Wu, T. F. Wenisch, and J. A. Fessler, "Iterative helical CT reconstruction in the cloud for ten dollars in five minutes," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2013, pp. 241–4. [Online]. Available: proc/13/web/rosen-13-ihc.pdf

[32] J. Cui, G. Pratx, B. Meng, and C. S. Levin, "Distributed MLEM: an iterative tomographic image reconstruction algorithm for distributed memory architectures," *IEEE Trans. Med. Imag.*, vol. 32, no. 5, pp. 957–67, May 2013.

[33] J. Xu, "Modeling and development of iterative reconstruction algorithms in emerging X-ray imaging technologies," Ph.D. dissertation, Washington University, St. Louis, May 2014. [Online]. Available: http://openscholarship.wustl.edu/etd/1270/

[34] M. Magnusson, P.-E. Danielsson, and J. Sunnegardh, "Handling of long objects in iterative improvement of nonexact reconstruction in helical cone-beam CT," *IEEE Trans. Med. Imag.*, vol. 25, no. 7, pp. 935–40, July 2006.

[35] W. P. Segars, M. Mahesh, T. J. Beck, E. C. Frey, and B. M. W. Tsui, "Realistic CT simulation using the 4D XCAT phantom," *Med. Phys.*, vol. 35, no. 8, pp. 3800–8, Aug. 2008.

[36] J. M. Borwein and Q. J. Zhu, *Techniques of variational analysis*. Springer, 2005.

[37] S. Boyd and L. Vandenberghe, *Convex optimization*. UK: Cambridge, 2004. [Online]. Available: http://www.stanford.edu/~boyd/cvxbook.html

Appendix A
Fenchel duality for GPU-based reconstruction algorithm

Proving (16) involves a straightforward application of Fenchel's duality theorem, see *e.g.*, [36, Theorem 4.4.3]. Define

$$f(\mathbf{x}) = \frac{\mu}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|_2^2, \tag{60}$$

$$\mathbf{K} = \begin{bmatrix} \mathbf{A} \\ \mathbf{C} \\ \mathbf{I} \end{bmatrix}. \tag{61}$$

We write the blocks of elements of $\mathbf{Kx}$ as $[\mathbf{Kx}]_{\mathbf{u}}$, $[\mathbf{Kx}]_{\mathbf{v}}$ and $[\mathbf{Kx}]_{\mathbf{z}}$. Define

$$g(\mathbf{Kx}) = \mathsf{L}([\mathbf{Kx}]_{\mathbf{u}}) + \mathsf{R}([\mathbf{Kx}]_{\mathbf{v}}) + \mathsf{N}([\mathbf{Kx}]_{\mathbf{z}}). \tag{62}$$

The value attained by the primal update problem (5), can be written in this terminology as

$$p = \min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{Kx}) = \min_{\mathbf{x}} \mathsf{J}^{(n)}(\mathbf{x}). \tag{63}$$

The convex conjugates of f and g are [37, pg. 95]

$$f^*(\mathbf{x}^*) = \frac{1}{2\mu}||\mathbf{x}^*||_2^2 + (\mathbf{x}^*)^\intercal \mathbf{x}^{(n)}, \tag{64}$$

$$g^*(\mathbf{q}^*) = \mathsf{L}^*(\mathbf{q}_{\mathbf{u}}^*) + \mathsf{R}^*(\mathbf{q}_{\mathbf{v}}^*) + \mathsf{N}^*(\mathbf{q}_{\mathbf{z}}^*). \tag{65}$$

The value attained by maximizing the dual function (19) is

$$d = \sup_{\mathbf{q}^*} -f^*(-\mathbf{K}^\intercal \mathbf{q}^*) - g^*(\mathbf{q}^*) = \sup_{\mathbf{q}^*} \mathsf{D}^{(n)}(\mathbf{q}_{\mathbf{u}}^*, \mathbf{q}_{\mathbf{v}}^*, \mathbf{q}_{\mathbf{z}}^*). \tag{66}$$

Note that although (66) apparently differs from the statement in [36, Theorem 4.4.2] by a sign, the expressions are equivalent.

The domain of f is dom $f = \mathbb{R}^N$, and the image of dom f under $\mathbf{K}$ is $\mathbf{K}$ dom f = range $\mathbf{K}$. The set over which g is continuous is cont $g = \{\boldsymbol{\theta} : \boldsymbol{\theta}_{\mathbf{z}} > \mathbf{0}\}$.

Finally, by the Fenchel duality theorem, because

$$\mathbf{K} \text{ dom } f \bigcap \text{cont } g \neq \emptyset, \tag{67}$$

and f and g are both convex functions, $p = d$.

Appendix B
Equivalence of primal- and dual-based solutions

Let the value of $\mathbf{x}^{(n+1)}$ produced by solving the primal update problem (5) be

$$\mathbf{x}_p = \operatorname*{argmin}_{\mathbf{x}} \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathbf{z}). \tag{68}$$

The value of $\mathbf{x}^{(n+1)}$ induced by solving the dual problem (19) is

$$\mathbf{x}_d = \widetilde{\mathbf{x}}^{(n+1)}\left(\widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right), \tag{69}$$

$$\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u}, \mathbf{v}, \mathbf{z}) = \operatorname*{argmin}_{\mathbf{x}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathbf{z}), \tag{70}$$

where

$$\begin{aligned} \widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)} &= \operatorname*{argmax}_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{D}^{(n)}(\mathbf{u}, \mathbf{v}, \mathbf{z}) \\ &= \mathsf{S}^{(n)}\left(\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u}, \mathbf{v}, \mathbf{z}), \mathbf{u}, \mathbf{v}, \mathbf{z}\right). \end{aligned} \tag{71}$$

Our goal is to show $\mathbf{x}_p = \mathbf{x}_d$.

We proceed by contradiction. Suppose $\mathbf{x}_p \neq \mathbf{x}_d$. Because $\mathsf{S}^{(n)}$ is strongly convex and $\mathbf{x}_d$ minimizes $\mathsf{S}^{(n)}$ when the dual variables are fixed at $\left(\widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right)$ (70),

$$\begin{aligned} d &= \mathsf{S}^{(n)}\left(\mathbf{x}_d, \widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right) \\ &< \mathsf{S}^{(n)}\left(\mathbf{x}_p, \widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right) \\ &\leq \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x}_p, \mathbf{u}, \mathbf{v}, \mathbf{z}) = p, \end{aligned} \tag{72}$$

contradicting $p = d$ (see Appendix A). Thus, $\mathbf{x}_p = \mathbf{x}_d$.

Appendix C
Convergence for GPU-based reconstruction algorithm with approximate updates

If the maximizing dual variables are found exactly (*i.e.*, if (20) holds with equality), then the proposed algorithm is a simple majorize-minimize procedure (5) and $\left\{\mathbf{x}^{(n)}\right\}$ converges to a minimizer of the cost function [24]. Finding the exact maximizers of $\mathsf{D}^{(n)}$ is too computationally expensive, so we settle for approximate optimization. Fortunately, under conditions similar to those for other approximate-update algorithms like ADMM [21], the proposed algorithm can converge even with inexact maximization of $\mathsf{D}^{(n)}$.

Let $\epsilon_{\mathbf{u}}^{(n+1)}$, $\epsilon_{\mathbf{v}}^{(n+1)}$ and $\epsilon_{\mathbf{z}}^{(n+1)}$ be the weighted error between the approximate maximizers $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ of $\mathsf{D}^{(n)}$ and the true maximizers $\widehat{\mathbf{u}}^{(n+1)}$, $\widehat{\mathbf{v}}^{(n+1)}$, $\widehat{\mathbf{z}}^{(n+1)}$:

$$\epsilon_{\mathbf{u}}^{(n)} = \left|\left|\widehat{\mathbf{u}}^{(n)} - \mathbf{u}^{(n)}\right|\right|_{\mathbf{AA}^\intercal}, \qquad \epsilon_{\mathbf{v}}^{(n)} = \left|\left|\widehat{\mathbf{v}}^{(n)} - \mathbf{v}^{(n)}\right|\right|_{\mathbf{CC}^\intercal},$$
$$\epsilon_{\mathbf{z}}^{(n)} = \left|\left|\widehat{\mathbf{z}}^{(n)} - \mathbf{z}^{(n)}\right|\right|. \tag{73}$$

Assume that we solve the dual maximization subproblem (20) well enough that these errors are summable:

$$\sum_{n=1}^{\infty} \epsilon_{\mathbf{v}}^{(n)} < \infty, \qquad \sum_{n=1}^{\infty} \epsilon_{\mathbf{u}}^{(n)} < \infty, \qquad \sum_{n=1}^{\infty} \epsilon_{\mathbf{z}}^{(n)} < \infty. \quad (74)$$

Let $\widehat{\mathbf{x}}^{(n+1)}$ be the exact solution to the primal update problem (5). The error between the approximate update $\mathbf{x}^{(n+1)}$ and the exact update, $\widehat{\mathbf{x}}^{(n+1)}$, is

$$\begin{aligned} \epsilon_{\mathbf{x}}^{(n)} &= \left|\left| \mathbf{x}^{(n+1)} - \widehat{\mathbf{x}}^{(n+1)} \right|\right| \\ &= \left|\left| \widetilde{\mathbf{x}}^{(n+1)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right) \right.\right. \\ &\qquad \left.\left. - \widetilde{\mathbf{x}}^{(n+1)}\left(\widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right) \right|\right| \\ &\leq \frac{1}{\mu}\left(\epsilon_{v}^{(n)} + \epsilon_{u}^{(n)} + \epsilon_{\mathbf{z}}^{(n)}\right), \end{aligned} \quad (75)$$

using the form of the dual-induced primal solution (17) and the triangle inequality. Because the dual update errors are summable (74), the primal update errors $\left\{\epsilon_{x}^{(n)}\right\}$ are also summable. Then, by [21, Theorem 3], the proposed algorithm is a convergent "generalized proximal point algorithm" and produces a sequence of iterates $\left\{\mathbf{x}^{(n)}\right\}$ that converge to a minimizer $\widehat{\mathbf{x}}$.

In practice, it may be difficult to verify numerically that the conditions (74) hold, but at least this analysis provides some sufficient conditions for convergence. In contrast, OS algorithms [12] have no convergence theory (and can diverge even for well-conditioned problems).

Alternating dual updates algorithm for X-ray CT reconstruction on the GPU: supplementary material

Madison G. McGaffin, *Member, IEEE,* and Jeffrey A. Fessler, *Fellow, IEEE*

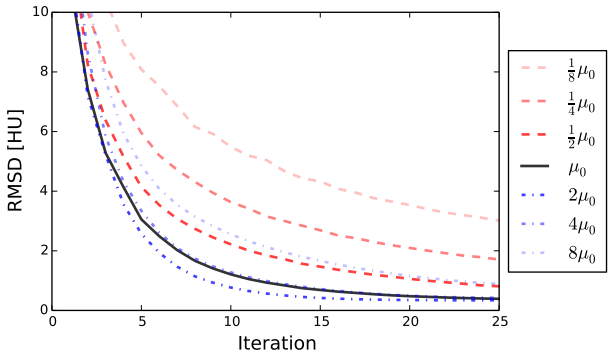


Fig. 7: Root mean squared difference (RMSD) to a converged reference *vs.* iteration for the helical shoulder scan for several values of the parameter μ around μ_0, the heuristic in (57).

This document provides supplementary experiments for the publication [1]. Section I contains an experiment modifying the proximal parameter μ around the heuristic in the paper (57). Section II contains results from an image reconstruction using a total-variation (TV)-like regularizer.

I. Varying the proximal penalty μ

The proximal parameter μ (6) is selected in [1] using a heuristic derived from the statistical weights $\mathbf{W}$ and the tomography majorizers $\{\mathbf{M}_g\}$ in (57). This heuristic was used for the reconstructions in Sections IV-C and IV-D.

We performed the helical patient data reconstruction in Section IV-C with several values of μ around μ_0, the heuristic in (57). Figure 7 shows the root mean squared difference (RMSD) of each algorithm as a function of iteration; changing μ does not change the value that each algorithm converges to, assuming the algorithm converges. Changing μ also does not change how long each iteration takes to perform.

It appears that the heuristic (57) is suboptimal, but it appears to be a good "default" setting.

II. Reconstruction with a TV-like penalty

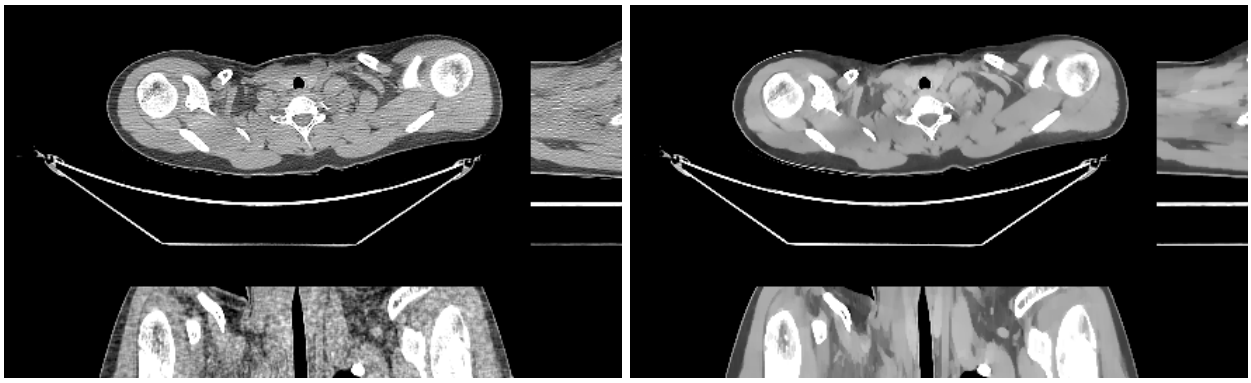
We altered the regularizer in Section IV-C to approximate absolute value potential function used in TV. We decreased the parameter δ of the Fair potential (39) from $\delta = 10$ to $\delta = 0.1$ and scaled up the regularizer weights $\{\beta_k\}$ by 100 times. Figure 8b shows a reference image using the new regularizer; compared to Figure 3b, the reconstruction has more uniform regions and some "cartoony" artifacts.

We ran OS-OGM and the proposed algorithm on an NVIDIA Kepler K5200; Figures 8c, 8d and 8e plot the RMSD of each algorithm to the reference as a function of equit and time. The lower value of δ results in a higher curvature regularizer majorizer for the OS-OGM algorithm that significantly slows its convergence. Because the proposed algorithm does not majorize the regularizer, it does not suffer from this slowdown. In fact, the proposed algorithm approaches the reference image more quickly in this experiment than in the results shown in Figure 4 of [1].

Supported in part by NIH grant U01 EB 018753, and by equipment donations from Intel Corporation.

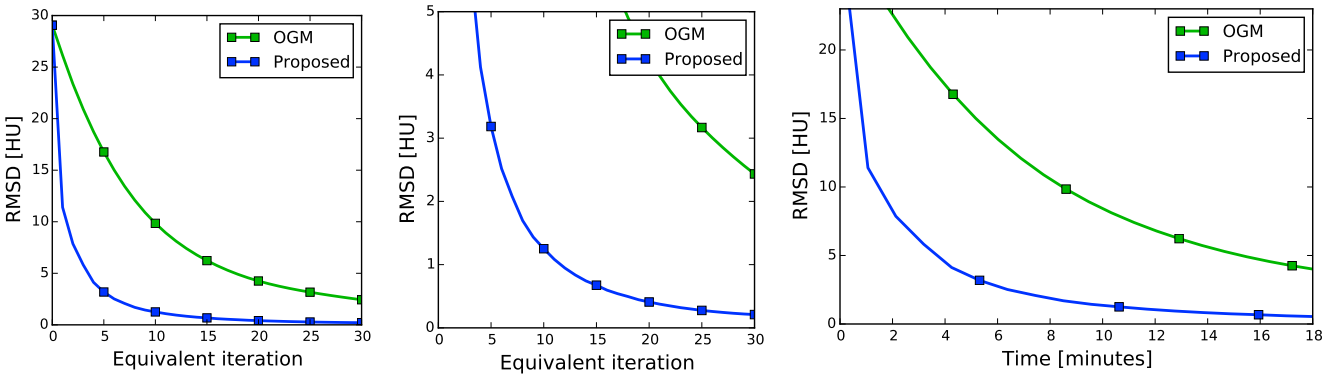
References

[1] M. McGaffin and J. A. Fessler, "Alternating dual updates algorithm for X-ray CT reconstruction on the GPU," *IEEE Trans. Computational Imaging*, 2015, to appear.



(a) Filtered backprojection

(b) Reference



(c) RMSD *vs.* equit

(d) RMSD *vs.* equit (zoomed)

(e) RMSD *vs.* time

Fig. 8: Image reconstruction of the helical shoulder case in Section IV-C using a TV-like Fair potential with parameter $\delta = 0.1$.