Journal of Signal Processing Systems (2019) 91:321–338
https://doi.org/10.1007/s11265-018-1352-0

GPU-Based Iterative Medical CT Image Reconstructions

Xiaodong Yu[1] · Hao Wang[1] · Wu-chun Feng[1] · Hao Gong[2] · Guohua Cao[2]

Received: 17 August 2017 / Revised: 8 December 2017 / Accepted: 13 February 2018 / Published online: 8 March 2018

Abstract
The algebraic reconstruction technique (ART) is an iterative algorithm for CT (i.e., computed tomography) image reconstruction that delivers better image quality with less radiation dosage than the industry-standard filtered back projection (FBP). However, the high computational cost of ART requires researchers to turn to high-performance computing to accelerate the algorithm. Alas, existing approaches for ART suffer from inefficient design of compressed data structures and computational kernels on GPUs. Thus, this paper presents our CUDA-based CT image reconstruction tool based on the algebraic reconstruction technique (ART) or cuART. It delivers a compression and parallelization solution for ART-based image reconstruction on GPUs. We address the under-performing, but popular, GPU libraries, e.g., cuSPARSE, BRC, and CSR5, on the ART algorithm and propose a symmetry-based CSR format (SCSR) to further compress the CSR data structure and optimize data access for both SpMV and SpMV_T via a column-indices permutation. We also propose sorting-based global-level and sorting-free view-level blocking techniques to optimize the kernel computation by leveraging different sparsity patterns of the system matrix. The end result is that cuART can reduce the memory footprint significantly and enable practical CT datasets to fit into a single GPU. The experimental results on a NVIDIA Tesla K80 GPU illustrate that our approach can achieve up to 6.8x, 7.2x, and 5.4x speedups over counterparts that use cuSPARSE, BRC, and CSR5, respectively.

Keywords GPU · Computed tomography · Image reconstruction · Algebraic reconstruction technique · Sparse matrix-vector multiplication · Transposed SpMV

1 Introduction and Motivation

The x-ray computed tomography (CT) scan is an indispensable medical imaging diagnostic tool. Its usage has sharply increased over the past few decades [1]. There are two categories of CT image reconstruction: analytical methods, e.g., filtered back projection (FBP), and iterative methods, e.g., algebraic reconstruction technique (ART) [2]. Currently, FBP is the industry standard due to its fast reconstruction speed. Although FBP can start reconstructing images once the first projection is acquired, it is sensitive to projection noise and requires more X-ray flux to deliver better reconstructed image quality. In contrast, ART is insensitive to the noise and can provide better image quality with fewer projection views and less radiation dosage. However, the usage of ART is hindered by its high computational cost, especially for clinical applications that require instantaneous image reconstruction. As a result, the acceleration of ART is of paramount importance.

Significant research effort has already been invested on the acceleration of ART, both algorithmically and at runtime. Some try to accelerate ART by modifying the algorithm [3–5], while others try to map ART to high-performance computing (HPC) platforms, e.g., multicore processors [6], Beowulf clusters [7], and FPGAs [8].

✉ Xiaodong Yu
xdyu@vt.edu

Hao Wang
hwang121@vt.edu

Wu-chun Feng
wfeng@vt.edu

Hao Gong
haog1@vt.edu

Guohua Cao
ghcao@vt.edu

1 Department of Computer Science, Virginia Tech, Blacksburg, VA, USA

2 Department of Biomedical Engineering and Mechanics, Virginia Tech, Blacksburg, VA, USA

Recently, GPUs have been employed as viable accelerators for ART-based CT image reconstruction [9–12], due to their superior energy efficiency and performance, e.g., kernel-only speedups of 10x to 15x over CPU-based solutions, when the compressed sparse row (CSR) format is used to compress data and advanced linear algebra subroutines (e.g., cuSPARSE [11]) are used to accelerate computations. However, a practical CT system matrix stored in the CSR format may still exceed a GPU's memory capacity. Moreover, applying the approaches found in existing libraries to ART may result in sub-optimal performance [13–15].

Our previous research [16, 17] propose the CUDA-accelerated algebraic reconstruction technique (i.e., cuART), which provides a complete compression and parallelization solution for ART-based CT image reconstruction on GPUs. The cuART includes our symmetry-based, CSR format (*SCSR*), two blocking techniques, parallelized ART and a transpose-free GPU kernel. The SCSR is a *CSR* variant dedicated for the *S*ystem matrix. It compresses data by leveraging CT physical characteristics. We also propose a column-indices permutation in SCSR in order to optimize data-access patterns for sparse matrix-vector (SpMV) multiplication and its transpose (SpMV_T). The blocking techniques in cuART transform the system matrix in SCSR format to several dense sub-matrices. We provide two blocking techniques leveraging two different levels of system matrix sparsities. The sorting-based global-level blocking leverages global-level sparsity pattern and is intuitive and easy to be implemented. On the other hand, the view-level blocking requires more a complicated transformation algorithm, but is sorting-free and converges faster due to leveraging finer grained (local) view-level sparsity pattern. Moreover, we address the data dependencies in the original ART algorithm and propose a parallelized ART. Though our parallelized ART slightly sacrifices convergence speed, we verify that it does not affect the reconstruction accuracy and can deliver overall speedup that surpasses the original ART algorithm. Based on these optimizations, we then provide a transpose-free GPU kernel in cuART to achieve significant performance improvements over other general SpMV schemes on GPUs.

This paper extends our previous work in four aspects: firstly, we add two subsections to provide elaborate analyses and rigorous mathematical proof on view-level sparsity characteristics. These characteristics are essential to view-level blocking. We also provide the detailed view-level blocking algorithm. Secondly, we thoroughly evaluate the quality of our reconstructed images using advanced statistical measurements (e.g SSIM, line profile). Thirdly, we discuss the future work regarding how our symmetry-based compression mutually benefits the Low-Dose CT, a new trend in medical imaging research. Finally, we add some new proposed researches to the related work to reflect the most recent advances in the community. Our major contributions include:

- A SCSR format that leverages the symmetries of a system matrix to further compress CSR-based data and optimize the data access for both SpMV and SpMV_T by permuting column indices.
- Two blocking techniques to convert SCSR to dense sub-matrices at runtime by leveraging different level sparsity patterns of the system matrix.[1]
- A parallelized ART algorithm, along with the design of a transpose-free GPU kernel.[2]
- A prototype and performance evaluation of our approach on the NVIDIA Kepler K80 GPU, which in turn, can achieve up to 6.8, 7.2, and 5.4-fold speedups over its counterparts that use cuSPARSE, BRC, and CSR5, respectively.

2 Background and Related Work

2.1 CT Rational & System Matrix

A CT scan constructs cross-sectional images of a scanned object, relying on a series of X-ray projection data. Figure 1 shows a schematic diagram of a spiral CT, the dominant structure of commodity CT scanners. For such a structure, the object is placed between a X-ray light source and detector. The X-ray source emits multiple rays that pass through the object, and the detector collects radiation that is not absorbed by the object. The X-ray source and detector spin around the central axis of the object with constant step length, and multiple one-dimensional (1D) projection data at varying views are collected.

According to Beer's law, the mathematical model of CT is a *Radon transform* of the object image to the projection data. It has a discrete form as a linear equation system: $\boldsymbol{WF}=\boldsymbol{P}$, where $\boldsymbol{F}$ is the image pixel vector of object image, $\boldsymbol{W}$ is the pre-computed system matrix, and $\boldsymbol{P}$ is the collected projection vector. Typically, the object image is a square with $N = n \times n$ pixels, in which the pixel values represent attenuations of tissue; accordingly, $\boldsymbol{F}$ is $N \times 1$. The X-ray light source emits l rays and each scan has v projection views, hence $\boldsymbol{P}$ is $M \times 1$, where $M = l \times v$. $\boldsymbol{W}$ is a $M \times N$ matrix that stores weighting factors; its element w_{ij} represents the contribution proportion of jth pixel to

[1]The sorting-based global-level blocking is easy to be implemented, while the sorting-free view-level blocking delivers faster preprocessing time and less data padding and can also enable the adapted algorithm to converge faster.

[2]This kernel leverages the merits of our SCSR format and blocking techniques to provide significant performance improvements.

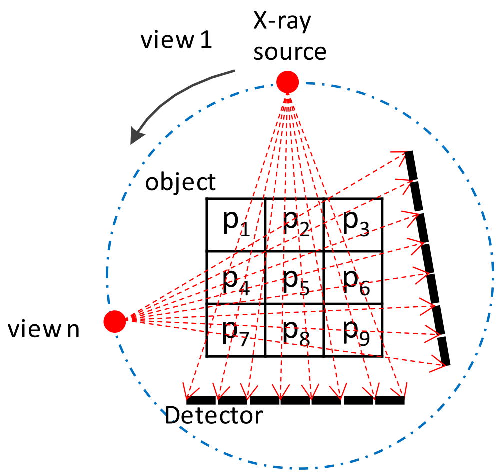


Figure 1 Schematic diagram of the spiral CT structure.

ith projection data. A real-world scenario would estimate a 1024×1024 image through a 1024×720 projection vector, then the corresponding system matrix would have $1024^3 \times 720$ elements and require several terabytes (TB) to be stored. Obviously, the approach that stores and utilizes the whole system matrix may dominate the reconstruction efficiency.

2.2 Algebraic Reconstruction Technique (ART)

CT image reconstruction is based on the *inverse Radon transform*. Such an inversion is practically impossible to be done via the analytical algebraic method [18]. So, Gordon et al. [2] proposed the algebra reconstruction technique (ART), which is an iterative solver for systems of linear equations. ART starts from an initial guess of an image vector $F^{(0)}$, then repeatedly updates the estimated image until its pixel values are convergent, according to some criteria. Each iteration of ART can be mathematically defined as follows:

$$f_j^{(i)} = f_j^{(i-1)} + \lambda \frac{p_i - \sum_{j=1}^{N}(f_j^{(i-1)} w_{ij})}{\sum_{j=1}^{N} w_{ij}^2} w_{ij} \tag{1}$$

where $i = 1, 2, ..., M$; $j = 1, 2, ..., N$; λ is the relaxation parameter; $f_j^{(i)}$ is jth pixel of the estimated image after i iterations ($f^{(0)}$ is an all-zero vector), p_i is ith element of the projection vector, and w_{ij} is the element of the system matrix at ith row and jth column that represents the weighting factor of the jth pixel to the ith projection data. Iterating i in Eq. 1 from 1 to M is one round of ART, in which the ith iteration will update the whole estimated image $f^{(i-1)}$ based on the corresponding projection data (p_i) and weighting factors (w_i). ART repeats until the estimated image F is convergent. Equation 1 shows that the heaviest computational burden of ART is the matrix-vector multiplication, i.e., $\sum_{j=1}^{N}(f_j^{(i-1)} w_{ij})$.

Two parallel variants, derived from ART, include (1) simultaneous iterative reconstruction technique (SIRT) [19] and (2) simultaneous algebraic reconstruction technique (SART) [20]. The former, i.e., SIRT, updates the estimated image after going through all the rays to process all system matrix rows at the same time (rather than using a ray-by-ray approach). However, SIRT takes longer to converge than ART. The latter, i.e., SART, is a tradeoff between ART and SIRT; it updates the image after going through rays within a certain angle, and hence, can simultaneously process system matrix rows that belong to the same view. SART converges faster than SIRT.

2.3 Sparse Matrix Compression and SpMV

As described in Section 2.1, storing the entire system matrix in a single GPU is infeasible. Existing proposals use dynamic approaches that calculate the system matrix on the fly [9, 10]. However, these approaches introduce large computational overhead due to repeated on-the-fly computations during iterative image reconstruction [12]. Fortunately, system matrices are sparse due to the physical characteristics of the weighting factors; hence, the design space for storage and computational optimizations can be explored.

Figure 2 shows the Area Integral Model (AIM) [5] of CT on a $n \times n$ object image. In this model, X-rays are considered as narrow fan-beams, and weighting factors are defined as the ratios of ray-pixel interaction area to pixel area. For example, the shaded area S_{ij} in Fig. 2 is the interaction area of ith ray r_i and jth pixel f_j. With the square pixel area being δ^2, hence the corresponding weighting factor is $w_{ij} = S_{ij}/\delta^2$. Intuitively, each ray just interacts with a few pixels, and only these interactions result in nonzero weighting

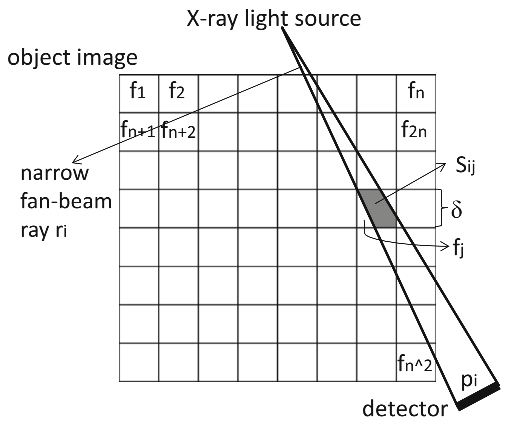


Figure 2 Fan-beam area integral model of CT.

factors. This means only a few elements of each system matrix row are nonzero, e.g., a system matrix density of only 10%. Hence, the matrix-vector multiplication from Eq. 1 falls into category of sparse matrix-vector multiplication (SpMV).

Sparse matrix compression and SpMV have been well studied over the past few decades. For example, the compressed sparse row (CSR) format is one of the most widely-used formats that compresses a matrix in row-major format and stores non-zero elements of the sparse matrix into contiguous memory locations. It consists of three vectors: *val*, *col_idx*, and *row_ptr*, which store non-zero elements, column indices, and offsets of rows, respectively. Figure 5a and b shows an system matrix and its compressed format in CSR. Although CSR is memory efficient because it stores no redundant elements, it is not computationally efficient because it requires indirect addressing step for each query.

In recent years, many CSR variants have been proposed. Blocked row-column (BRC) [13] scans the sparse matrix in two dimensions and groups neighboring rows to the several small blocks, in order to better balance the workload and reduce the amount of padding needed. CSR5 [15] evenly partitions non-zero elements into multiple 2D tiles to achieve load balance and avoid storing unnecessary data; and its extension [21] directly uses the CSR format for load balanced SpMV and also considers CPU-GPU heterogeneous processors. Compressed sparse blocks (CSB) [14] partitions a matrix into multiple equal-sized blocks and stores the non-zero elements within each block using Morton order; it can support equally efficient SpMV and SpMV_T using a uniform format. yaSpMV [22] provides an efficient segmented sum algorithm for the blocked compressed common coordinate (BCCOO) format with an auto-tuning framework for parameter selection. ACSR [23] leverages binning and dynamic parallelism techniques provided by modern GPUs to reduce thread divergence for the standard CSR. CSR-Adaptive [24] uses local scratchpad memory and dynamically assigns various number of rows to GPU compute units. More recently, NVIDIA proposed a merge-based parallel SpMV [25], which uses CSR to minimize preprocessing overhead and applies equitable multi-partitioning on the inputs to ensure load balance. Steinberger et al. [26] present an uncompromising approach for GPU-based SpMV. Their approach has no preprocessing overhead and is directly applicable to standard CSR. It exhibits efficient memory access patterns and steady per-thread workload from a global perspective, and avoids divergence execution from a local perspective by using an efficient encoding to reduce on-chip temporary data. It achieves 20% average improvement against CSR, and can significant benefit SpMV_T as well with the adaption. Hou et al. [27] propose a machine learning-based auto-tuning framework, to determine the optimal binning schemes and kernel for parallelized CSR-based SpMV. Although these proposals leverage the hardware features of the CPU or GPU to improve SpMV performance, they do not take any specific sparsity structures into consideration. Furthermore, a recent study indicates that sparse matrix transposition (SpMV_T) may become a performance bottleneck, especially when SpMV has been highly optimized [28]. Thus, transpose-free methods for SpMV and SpMV_T are in urgent need.

2.4 GPU-Based CT Image Reconstruction

Nowadays, researchers invest large effort on accelerating domain applications using High-Performance Computing platforms [29–32] in addition to mathematically optimizing the algorithms [33, 34]. GPU, one of the most popular HPC devices, has been used to accelerate a myriad of applications and achieved significant performance improvements [35–39]. For CT image reconstruction, many researchers have also looked into how to leverage GPUs. For example, Keck et al. [40] propose an efficient implementation of SART on CUDA-enabled GPU. Pang et al. [9] propose a ray-voxel hybrid driven method to map SART onto GPU architecture, while Zhao et al. [10] propose a CUDA-based GPU approach that includes empty-space skipping and a multi-resolution technique to accelerate ART. However, both approaches require a significant amount of memory due to their lack of awareness of the sparse algebra computational pattern and the need to have to calculate weighting factors on the fly. Such on-the-fly calculation wastes the computational resources. Guo et al. [12] propose a stored system matrix (SSM) algorithm to reduce the size of the system matrix in GPU memory by leveraging shift invariance for projection and backprojection under a rotating coordinate. However, this approach does not optimize the kernel performance on the GPU. Liu et al. [11] accelerate SART with cuSPARSE [41] after compressing the system matrix into CSR. They report their GPU implementation achieves around a 10-fold computational-kernel-only speedup over its single-threaded CPU counterpart, which is much lower than the cuSPARSE speedups claimed for other sparse applications. As a result, there exist potential opportunities to optimize such an application on the GPU by leveraging CT-specific characteristics. Similar strategy is applied to accelerating other applications on GPUs: for example, Aktulga et al. [42] accelerate nuclear configuration interaction calculations on GPUs through improving both SpMM and SpMM_T by leveraging domain-specific characteristics. In the remaining sections of this paper, we utilize the NVIDIA GPU architecture and its Compute Unified Device Architecture (CUDA) [43] programming model.

3 SCSR Format for System Matrix

The first and foremost challenge for cuART is how to efficiently store non-zero elements of the system matrix. Even though CSR is one of the most efficient compressed formats for a sparse matrix, the data size of the CSR-based CT system matrix may still be much larger than the memory capacity of a commodity GPU. The third row of Table 1 shows the sizes of some real-world system matrices in CSR format. A CSR format system matrix for a fine image, e.g., 720 views and 2048^2 pixels, can exceed the memory capacity of the commodity NVIDIA Tesla K80 GPU, which has 24GB global memory. Although partitioning the rows of the system matrix into multiple groups can alleviate the memory pressure, it introduces additional data-transfer overhead. Moreover, the ART algorithm requires both SpMV and SpMV_T, and transposing the matrix at runtime could incur signficant overhead.

In this section, we propose SCSR, a variant of CSR format that is dedicated to the CT system matrix. SCSR leverages the symmetries in the system matrix in order to further compress CSR data and properly permute the column indices to avoid matrix transpositions during kernel computation in cuART.

3.1 Symmetry-Based System Matrix Compression

CSR format already removes all zero elements of the system matrix but still can't fit into GPU global memory. This fact motivates us to exploit another compression method in addition to the data structure level compression. specifically, we try to reduce the number of storing necessary elements of the system matrix by leveraging physical characteristics of CT scan.

As introduced in the background, the weighting factors do not relate to pixel indices; they only correlate to the interacted areas of X-rays and pixels. Accordingly, two symmetric X-rays can have symmetric weighting factors. Weighting-factor symmetries appear as the same numerical values at symmetric pixel positions. We find that X-rays can have two kinds of symmetry: *reflection symmetry* and *rotational symmetry*. Figure 3 demonstrates the symmetries among views (X-rays): X-rays from view v_1 and view v_3 visually have reflection symmetry, while X-rays from view v_2 and v_4 have rotational symmetry. Specifically, assuming each view has m X-rays, if ray r_i in view v_1 has weighting factor value $|w|$ for pixel I_1, then ray $r_{(m-i)}$ in view v_3 should have the same weighting factor value $|w|$ for pixel I_2; meanwhile, if ray r_i in view v_2 has weighting factor value $|w'|$ for pixel I_1, then ray r_i in view v_4 should have the same weighting factor value $|w'|$ for pixel I_3.

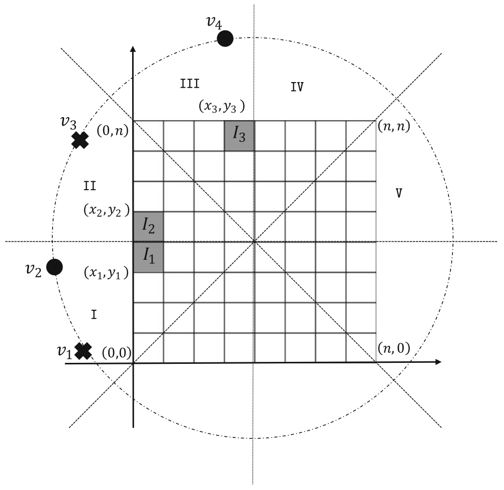


Figure 3 Schematic diagram of views/X-rays symmetries.

Based upon these symmetry characteristics, we propose a methodology to reduce the number of elements that have to be stored in in the system matrix in order to further compress the CSR data. Specifically, we equally divide the working area into eight zones (or eight sectors, as in Fig. 3). Only weighting factors coming from the views within one zone need to be stored; all the other weighting factors can then be easily obtained on the fly via simple index mappings. Figure 3 shows a coordinate system for the CT scan model on a $n \times n$ object image. Obviously, rays from zone II are reflection symmetric to their corresponding rays from zone I, while rays from zone III are rotationally symmetric to rays from zone I. A pixel at coordinate (x_1, y_1) (we denote its pixel index as $I_1 = n * y_1 + x_1$) should have corresponding pixels at symmetric positions; the indices of these corresponding pixels can be obtained through the following mapping rule:

$$\begin{cases} I_2 = n * y_2 + x_2 = n * (n - y_1) + x_1 \text{: reflection symm} \\ I_3 = n * y_3 + x_3 = n * (n - x_1) + y_1 \text{: rotational symm} \end{cases} \tag{2}$$

Table 1 Size comparison between the CSR-based and SCSR-based compressed system matrices.

projection_view#	360			720		
image_size (pixel#)	512^2	1024^2	2048^2	512^2	1024^2	2048^2
CSR-based matrix_size (GB)	2.34	6.19	18.44	4.82	11.97	37.41
SCSR-based matrix_size (GB)	0.29	0.77	2.31	0.60	1.50	4.68

Based on the analyses in the last paragraph, the weighting factors determined by a ray in zone I on pixel I_1 and its rotational symmetric ray in zone II on pixel I_2 are numerically identical. Similarly, a ray in zone I with pixel I_1 and its rotationally symmetric ray in zone III with pixel I_3 determine weighting factors having identical numerical value as well. Hence we only need to store the weighting factors (i.e., corresponding rows of the system matrix) coming from rays within zone I. Then during runtime, we can restore the weighting factors (i.e., rows) coming from rays in zone II and III via the stored rows by using the rule described in Eq. 2. We can restore rows from zone IV and V via rows from zone III using the same method, after clockwise rotating the coordinate system 90°. We can also analogously restore rows from all other zones. Applying this symmetry-based compression to CSR data can further compress it to one eighth of its original size. The forth row of Table 1 shows the data sizes of some SCSR-based system matrices. Compared to data sizes in third row, SCSR can fit into the global memory of a single GPU even in our largest case.

3.2 Column Indices Based Permutation for SCSR

As shown in Eq. 1, ART iteration requires multiplications of system matrix W and its transposition with the vector. One approach to process SpMV and SpMV_T is to transpose W to W^T in advance. However, this increases the memory footprint dramatically, especially given the fact that the system matrix needs several GB of memory even in SCSR format. Another method is to transpose the compressed matrix at the runtime, however, that significantly increases the computation time if multiple iterations are needed. An advanced data structure, CSB [14], proposes an uniform compressed matrix format to support fast SpMV and SpMV_T at the same time. However, it is not efficient to apply CSB on the system matrix for ART for two reasons: (1) Transforming the original system matrix to CSB format will break the index mapping rule in Eq. 2 that we use to compress the symmetric data and also destroy the sparse patterns that we use to generate dense sub-matrices (as discussed in the next section); (2) CSB is reported to be efficient on multi-core CPU but inefficient on GPU due to the overhead of handling divergent branches for dynamic parallelism and binary search [44].

Instead we propose an optimization using column indices based permutation, that is compatible to SCSR and can support both SpMV and SpMV_T at the same time. It is transpose-free and will not degrade cuART kernel performance. The conventional CSR-based transpose-free SpMV_T has to accumulate element-wise products to vector y. This approach requires atomic operations when multiple threads operate on the same y's element; otherwise, it results in a race condition. For example, in Fig. 5c, the last two elements in the second column with "4" ("4" is the column index in original SM) have element-wise products that need to be accumulated to the resultant vector element y_4 at the same time, which can result in a race condition. In order to avoid using atomic operations, we propose the column indices based permutation. Specifically, as shown in Fig. 5d and (e), we permute nonzero elements in each row in a round-robin manner to make elements in the same (SCSR) column have different (original SM) column indices. For example, elements in last two rows are rearranged in order to change their (original SM) column indices sequences from "3456" to "5364" and from "3456" to "6435" respectively. After such permutation, race condition will never happen during SpMV_T, since different threads always accumulate their products to different resultant vector elements at the same point in time. On the other hand, since elements are permuted inside each row, this method will not affect the correctness and efficiency of SpMV, as well as the matrix symmetry and sparsity.

4 Blocking Techniques for Computation Optimization

The distribution of nonzero elements along system matrix rows is irregular. In our test cases, the longest CSR row can be four times longer than the shortest one. Hence directly apply either CSR or our SCSR format to cuART can result in a lower performance than the expectation, due to threads workload imbalance. In this section, we propose two blocking techniques to re-organize SCSR data into multiple dense sub-matrices by utilizing system matrix sparsity patterns in different granularities.

4.1 Global-Level Blocking

Inspired by SIRT [19], we know simultaneously processing the whole system matrix is feasible. In this subsection, we provide a sorting-based blocking approach by leveraging the global-level sparsity pattern of system matrix. Figure 4 shows the methodology. According to the physical interpretation of system matrix, after shifting all its nonzero elements to the left, it has a *global sparsity pattern* as shown in (a). We can make two observations about this pattern: 1) number of nonzero elements(nnz) of each row falls into a narrow range; 2) the average of these nnz is close to the median of such range. Accordingly, after sorting the rows by nnz, the nonzero elements chunk will have a shape as shown in (b). Such shape can approximate to a right trapezoid as shown in (c). This trapezoid can then be transformed to a rectangle (dense block) by cutting the acute

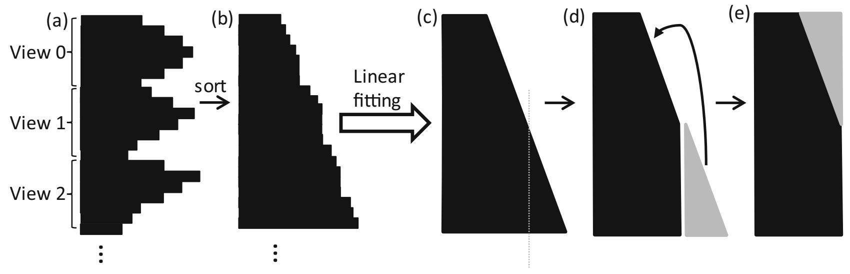


Figure 4 Schematic diagram for the methodology of leveraging the *global sparsity pattern* to block the SCSR data.

angle along a proper vertical line and rotating the cut off triangle 180° to fill the top-right indentation, as illustrated in (c)–(e).

Following the methodology, we provide the *global-level blocking* technique to transform CSR data to dense sub-matrices, as illustrated in Fig. 5. Figure 5a and b are the layouts of system matrix in original representation and CSR format respectively. Figure 5c shows the rearranged layout of the CSR-based representation, after all nonzero elements have been shifted to the left and the rows have been sorted based on their nnz. We pair the sorted rows following this rule: pair the first row to the last row, the second row to the second last row and so on. Then we concatenate and bisect each pair and pad them with zeros to get the dense matrix as shown in Fig. 5d. After permuting (as introduced in Section 3.2) the nonzero elements in (d) to (e), we divide the matrix in (e) (assuming the size is $M \times N$) to three dense sub-matrices as shown in Fig. 5f: Sub_SM_1 (with size $M \times N'$, where N' is the length of shortest row in (c)), Sub_SM_2, and Sub_SM_3 (both with size $\frac{M}{2} \times (N - N')$). Only elements of each Sub_SM_2 row may come from two different rows in (c). In order to record the junction point in each Sub_SM_2 row, we use an auxiliary bitmask matrix, in which 1s indicate junction point positions. For example, in the second row of Sub_SM_2 in (f), element with "1" comes from the second row in (c) while elements with "7" and "8" come from the ninth row in (c), hence the corresponding bitmask is 010.

Compared to BRC and CSR5, our global-level blocking scheme minimizes atomic operation penalty, thanks to the

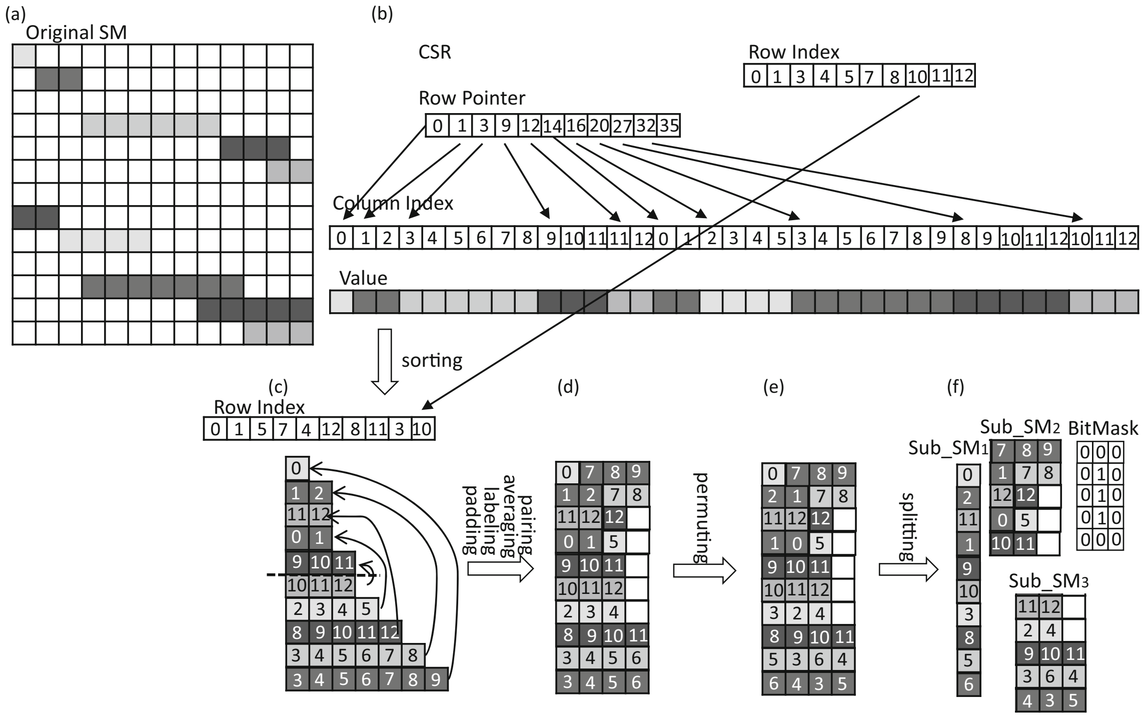


Figure 5 Schematic diagram of CSR to dense matrix based on global sparsity pattern of CT system matrix. White elements are zero paddings, numbers are column indices.

pairing technique we used that always groups nonzero elements into three dense sub matrices. Our proposed scheme also has no additional addressing step since the compressed rows can be directly mapped from row indices of the original system matrix. Last but not least, it only has one third of the sub-matrices (i.e., Sub_SM_2) that may cause slight divergences, since there is only at most one junction point in each row of that sub-matrix.

4.2 View-Level Blocking

Although our global-level blocking is easy to be implemented and works better than existing general blocking techniques, it remains two drawbacks: *a*) the heavy preprocessing overhead due to the sorting step, which may become a performance bottleneck of cuART if the system matrix is refreshed frequently. *b*) the slow convergence speed due to the compromised averaging approach on intermediate results. cuART with global-level blocking accumulates all intermediate results and averages the accumulation after processing all rays. This approach does not naturally comply with the principle of original ART algorithm, hence degrades convergence speed. Intrinsically, in the ART, projection data from one view alone can estimate the whole object image. As a result, different views can provide different estimations of the object image. Hence a more natural averaging approach should accumulate and average intermediate estimation results right after processing the rays within each view.

SART [20] approves in theory the feasibility of simultaneously processing the projection data and corresponding system matrix rows within one view. Accordingly, we propose a *view-level blocking* scheme, that is *sorting-free* and based on *view-level sparsity patterns* of the system matrix.

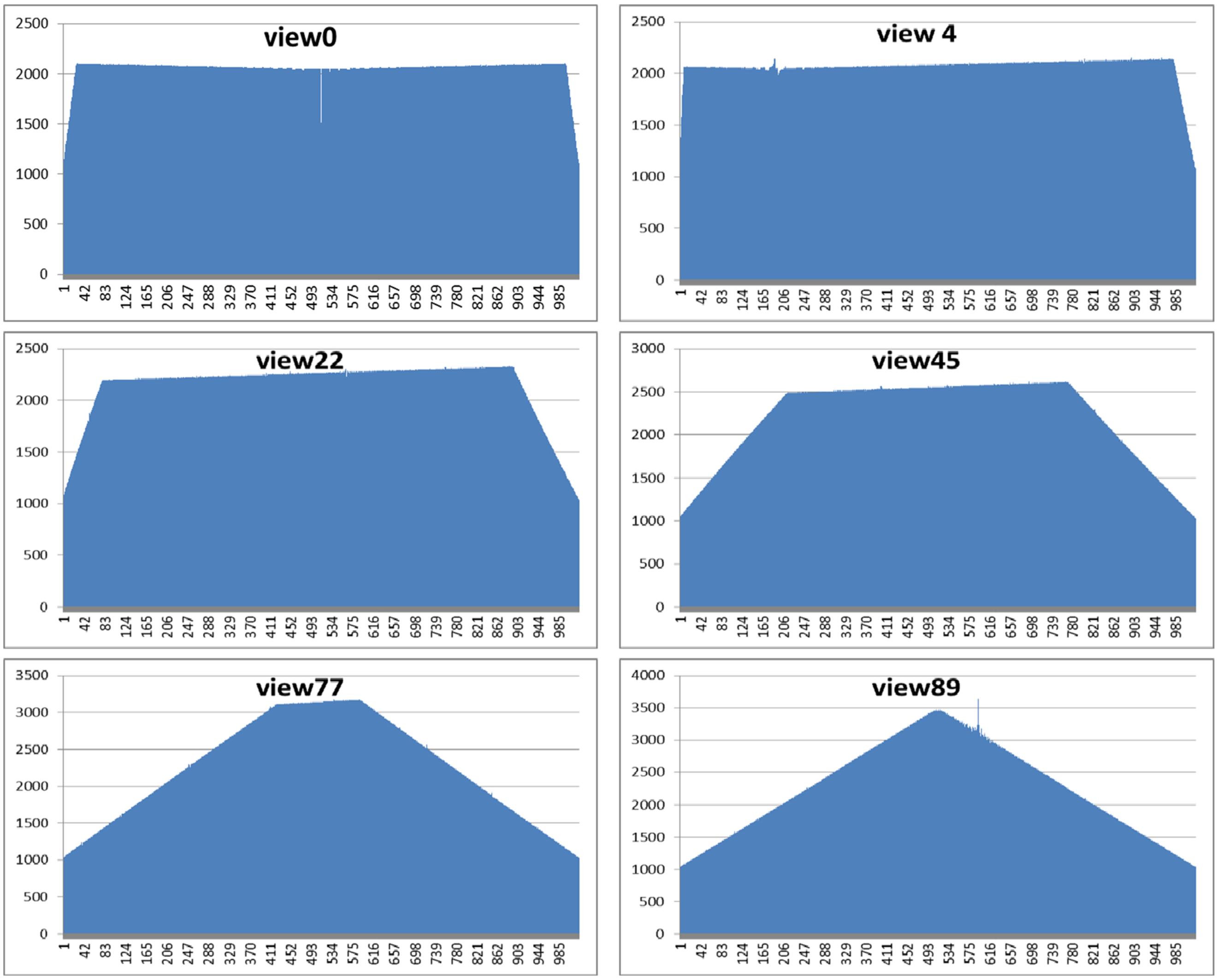


Figure 6 Sparsities of six typical projection views.

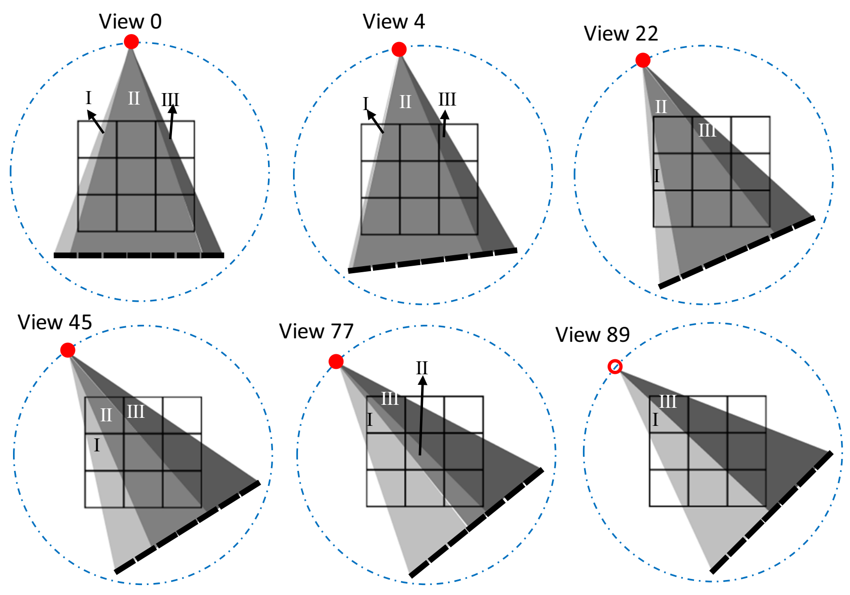


Figure 7 Schematic diagrams of view positions and shapes of projection/image overlap.

4.2.1 Overview of the View-level Sparsity

We start the introduction from the illustration of view-level sparsity patterns. As introduced in the last section, after applying symmetry-based compression, only 90 out of 720 views are demanded to be stored. Figure 6 shows the sparsities of six typical views in the case of 1024 rays per view with 1024^2 image. The characteristics of view-level sparsity patterns can be summarized as: *a*) most views' smallest row nnz are similar (around 1000 in example case), and largest row nnz are progressive (gradually increase from 2000 to 3500 in example case); *b*) each view pattern has a steady area in the middle with left/right linear increasing/decreasing areas in the sides respectively; *c*) the slopes of right areas increase gradually along views change, while the slopes of left areas first decrease then increase.

These characteristics are determined by the shape of overlaps between the X-rays and the object image at each view position. Figure 7 shows the corresponding view positions and overlap shapes of six typical views in Fig. 6. The overlapped region on each view is split to several sub-regions by the rays passing through image vertices. Specifically, view0 and view4 are divided to three sub-regions by the rays passing through two lower vertices. Along with light source moving anticlockwise, region I of the views go smaller. After a certain view, region Is will no longer be determined by the ray going through lower-left vertex; instead, the ray going through upper-left vertex will partition the region Is of the following views such as view22, view 45 and view77. Region Is in these views go larger along with light source moving anticlockwise. Finally in view 89, the ray passing through upper-left vertex will also go through lower-right vertex; consequently, region II is squashed and only two sub-regions: I and III are left. We can also observe that region IIs go smaller and region IIIs go larger all the way from view0 to view 89. The region Is map to the left trapezoids, the region IIIs map to the right trapezoids, and the region IIs result in the steady areas in Fig. 6. In other words, two inflection points on each shape are determined by two separator rays for each view. Accordingly, since the physical interpretation of the nnz of each row is the number of pixels overlapped with the corresponding ray, the largest and smallest row nnz of each view are determined by the lengths of separator rays (i.e. the line segments overlapping with the object image); largest nnz of the views go larger along with light source moving anticlockwise, while smallest nnz can keep approximately the same.

4.2.2 Mathematical Analyses of the View-Level Sparsity Characteristics

The above observed characteristics are not specific enough to make the view-level sparsity applicable to the blocking scheme design. In order to leverage the view-level sparsity patterns, we reveal three non-intuitive characteristics through mathematical analyses: a) positions of inflection points on shapes from adjacent views have constant distance, *b*) view 5 is the border-view that its left area of sparsity pattern has the valley slope (refer to characteristic (c) in first paragraph of section 4.2.1), and *c*) the right inflection points move right with five rows/step along views; the left inflection points first move left with five rows/step, then after view 5, they start to move right with six rows/step.

Figure 8 shows a sample view V, in which x is the light source, y is the center of both the image and virtual circle

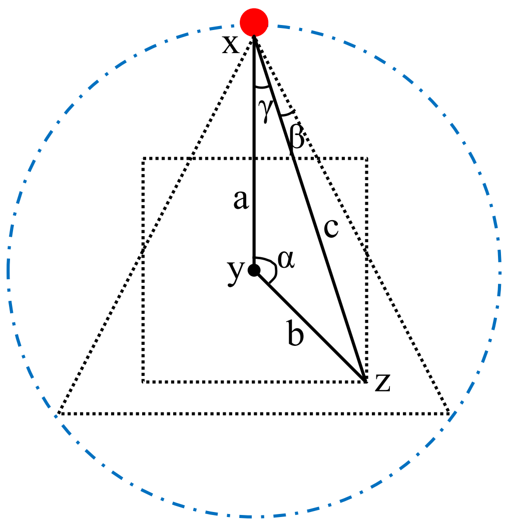


Figure 8 Schematic diagram of a sample view V.

and on the midline of projection triangle, and z is lower-right vertex of image. Edge xy, yz and xz have length a, b and c respectively. α, β, γ are three angles as indicated in figure; exploring how right inflection point position changes between different views is equivalent to understanding how $\angle\beta$ changes. Since $\angle\beta + \angle\gamma$ is fixed, we can analyze how $\angle\gamma$ changes instead.

According to the *law of cosine*, the edge lengths satisfy equations:

$$\begin{cases} c^2 = a^2 + b^2 - 2ab\cos\alpha \\ b^2 = a^2 + c^2 - 2ac\cos\gamma \end{cases} \tag{3}$$

After applying substitutions to them, we have

$$\gamma = \arccos\left(\frac{a - b\cos\alpha}{\sqrt{a^2 + b^2 - 2ab\cos\alpha}}\right) \tag{4}$$

According to the differential formula

$$\mathrm{d}(\arccos \mathrm{x}) = \frac{1}{\sqrt{1 - x^2}}\mathrm{d}x \tag{5}$$

if $\angle\gamma'$ is angularly similar enough to $\angle\gamma$, there is an approximate equation:

$$\frac{\arccos\gamma - \arccos\gamma'}{\gamma - \gamma'} \approx \frac{1}{\sqrt{1 - \gamma^2}} \tag{6}$$

Assuming $\angle\gamma'$ is the angle at the same position in view V' which is adjacent to V, we substitute Eq. 4 into Eq. 6 and get:

$$\gamma - \gamma' = \frac{\frac{a-b\cos\alpha}{\sqrt{a^2+b^2-2ab\cos\alpha}} - \frac{a-b\cos\alpha'}{\sqrt{a^2+b^2-2ab\cos\alpha'}}}{\sqrt{1 - \left(\frac{a-b\cos\alpha}{\sqrt{a^2+b^2-2ab\cos\alpha}}\right)^2}} \tag{7}$$

In practice, the X-ray light source moves with 0.5° angular step i.e. $\angle\alpha' = \angle\alpha + 0.5°$; accordingly, the numerical values of $\cos\alpha'$ and $\cos\alpha$ are close enough. a and b are fixed number once CT device is set, and won't change much even if device's physical parameters changing. Consequently, we approximately simplify Eq. 7 and get:

$$\begin{aligned} \Delta\gamma &\approx \frac{b(\cos\alpha' - \cos\alpha)}{\sqrt{a^2+b^2-2ab\cos\alpha-(a-b\cos\alpha)^2}} \\ &= \frac{(\cos\alpha' - \cos\alpha)}{\sqrt{1-\cos^2\alpha}} \approx \cot\alpha' - \cot\alpha \end{aligned} \tag{8}$$

Equation 8 shows, although $\Delta\gamma$ is not constant, it can be reasonably approximated to a fixed number due to the fact that cot is roughly linear within $[\frac{3\pi}{2}, 2\pi]$ interval. The left inflection point change can by analyzed analogously and lead to similar results. Through these analyses, we discover that inflection point positions change with constant row interval along views. In our case, the right inflection points move right with five rows per step along views; the left inflection points first move left with five rows per step; then after view 5, they start to move right with six rows per step.

4.2.3 View-Level Blocking Design

Based on all above analyses, we propose the *view-level blocking* that transforms SCSR to dense sub-matrices at runtime. Figure 9 shows the schematic diagram for the view-level sparsities based transformation. Algorithm 1 is the concrete method of the transformation using view-level blocking. Such blocking scheme has four steps: (a) Since all views have a similar smallest nnz, we calculate a split point for each view to divide the non-zeros to a dense matrix and a trapezoid (line 3-7 in algorithm 1). (b) Based on the fact that two neighboring views have nearly the same sparsity patterns, we pair adjacent views then cut and combine the left and right side triangles of the trapezoids along inflection point to transform them to rectangles (line 8-29; according to first paragraph of Section 4.2.2, the border-view is view 5, the c is 5 and the c′ is 6). Since inflection point positions change regularly, they can be directly addressed without sorting. When SpMV kernels access these dense rectangles, they can map their elements to the original system matrix without any indirect addressing. (c) Since the largest row nnz of each view gradually increases, we then average row lengths of each view pair (e.g. {view*i*, view(*89-i-1*)}, {view(*i+1*), view(*89-i*)}, where *i* is an integer less than 45) by moving elements and add flags (line 30–33). (d) We finally pad, combine the rows and generate three dense blocks for every two view pairs: sub_SM_1 that has $4M'$ rows without flags, sub_SM_2 that has $2M'$ rows with at most four flags per row, and sub_SM_3 that has less than $2M'$ rows with at most one flag per row, where M' is the number of system matrix rows for each view (line 34).

Algorithm 1 View-Level blocking technique

```
Function local_blocking(SM, nv, nr)
    ▷ SM is the system matrix; nv is # of views; nr is # of rays per view
    for $i \longleftarrow 0$ to $nv*nr-1$ do
        shift non-zeros of SM[i] to left;
        sub_SM_1[i] ← SM[i][0] to SM[i][a−1];   ▷ $a$ is a constant
    end
    SM ← (SM − sub_SM_1);
    for $i \longleftarrow 0$ to $nv-1$ stride 2 do
        /*left side triangles*/
        if $i<bv$ then
                ▷ $bv$ is the index of border-view
            for $j \longleftarrow 0$ to $b-i*c-1$ do
                ▷ $b$ is the inital # of rays in both size triangles; $c$ is the # of ray diff. between two consecutive views
                $SM^{(i)}[j] \leftarrow SM^{(i)}[j] + SM^{(i+1)}[b-i*c-1-j]$;
                delete $SM^{(i+1)}[b-i*c-1-j]$;
            end
        else
            for $j \longleftarrow 0$ to $(i-bv)*c'-1$ do
                ▷ $c'$ is the # of ray diff. after border-view
                $SM^{(i)}[j] \leftarrow SM^{(i)}[j] + SM^{(i+1)}[(i-bv)*c'-1-j]$;
                delete $SM^{(i+1)}[(i-bv)*c'-1-j]$;
            end
        end
        /*right side triangles*/
        for $j \longleftarrow 0$ to $b+i*c-1$ do
            $SM^{(i)}[nr-j] \leftarrow SM^{(i)}[nr-j] + SM^{(i+1)}[nr-(b+i*c-1)+j]$;
            delete $SM^{(i+1)}[nr-(b+i*c-1)+j]$;
        end
    end
    for $i \longleftarrow 0$ to $nv/2-1$ stride 2 do
        $sub\_SM_2 \leftarrow$ Pad(Avg($SM^{(i)}$, $SM^{((nv-1)-(i+1))}$));
        $sub\_SM_3 \leftarrow$ Pad(Avg($SM^{(i+1)}$, $SM^{((nv-1)-i)}$));
    end
    return $sub\_SM_1$, $sub\_SM_2$, $sub\_SM_3$
```

Compared to the global-level blocking, the view-level blocking may have slightly more thread divergences due to more flags and lower occupancy (sub_SM_3 has less than $2M'$ rows) in each single iteration. However, it has much less padding data than the global-level blocking due to the utilization of finer-grained sparsities. Moreover, cuART with view-level blocking can converge faster than with global-level blocking since the former updates the estimated image more frequently.

5 GPU-Based ART

5.1 Data Dependencies Decoupling

According to Eq. 1, each ART iteration starts with the estimated image updated by previous iterations. It updates the image using corresponding element of the projection vector and row of the system matrix. This implies that any two ART iterations have data dependency, and the system matrix rows should be accessed sequentially. On the other hand, both SIRT and SART suggest that system matrix rows can somehow be simultaneously processed. In our GPU-based design, we first decouple the data dependencies and rewrite the ART iteration as:

$$F^{(r+1)} = F^{(r)} + W^T(C.*(P - WF^{(r)})) \tag{9}$$

where F is $N \times 1$ pixel vector of the estimated image, its superscript r represents the index of parallelized ART round; W is $M \times N$ system matrix; P is $M \times 1$ projection

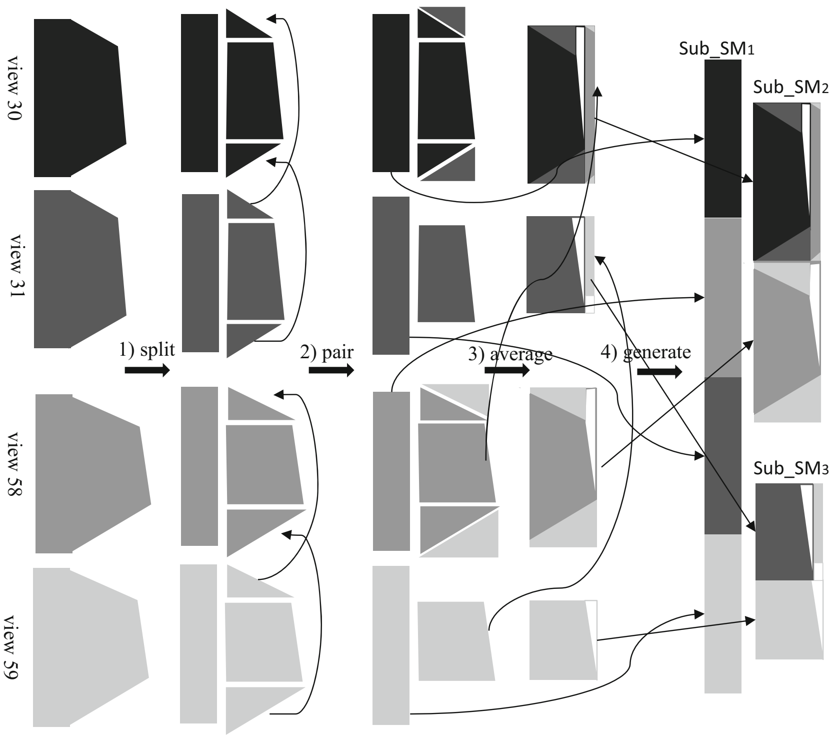


Figure 9 Schematic diagram of sample view for mathematical analysis. White areas are zero paddings.

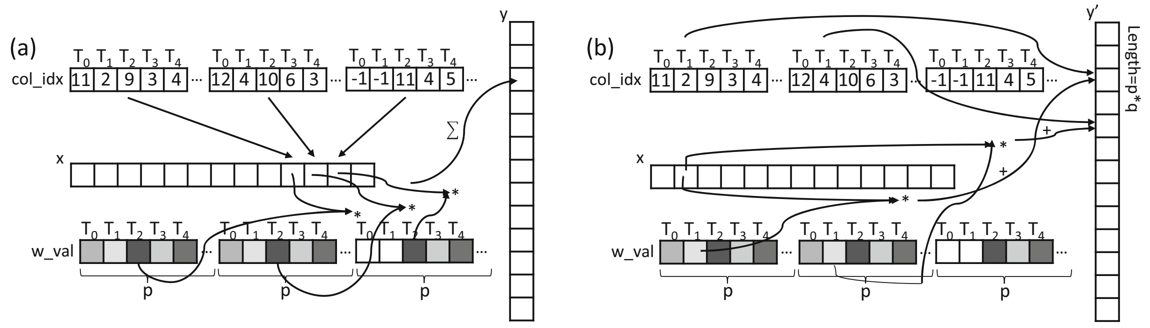


Figure 10 Matrix memory layout and **a** SpMV data access pattern and **b** SpMV_T data access pattern.

vector; C is $M \times 1$ constant vector whose elements are $c_i = ^{\lambda} / _{(M \sum_{j=1}^{N} w_{ij}^2)}$, where w_{ij} are system matrix elements; W^T is the transpose of system matrix and $.*$ is element-wise multiplication. Notice that W could be either the whole (with global-level blocking) or a portion (with view-level blocking) of the system matrix; the portion should be the rows that belong to the same view. In the latter case, all M should change to $M' = M/v$, where v is the number of views; accordingly, W has the size $M' \times N$, and P and C should be $M' \times 1$ vectors. Guan and Gordon [3] prove that the access order of X-rays and projection data change will not affect the reconstructed image quality that ART can achieve. This is because X-rays are not overlapped and each projection data just corresponds to a single ray. Hence, Eq. 9 can achieve the same reconstructed image quality to Eq. 1 after running adequate iterations. Although Eq. 9 has slower convergence speed than Eq. 1, our experiments show that the performance improvement getting from parallelization on GPU can offset the overhead caused by the slower convergence speed.

5.2 GPU Kernel of cuART

The GPU kernel of our cuART could support both SpMV and SpMV_T at the same time by using the SCSR based system matrix with our blocking techniques, and is transpose-free and race-free. Algorithm 2 shows such kernel, and Fig. 10 shows memory layout of system matrix (using one dense sub-matrix as example) and data access pattern.

Specifically, Fig. 10a illustrates the memory layout of an example dense sub-matrix Sub_SM_3 (with the size p * q) and the data access pattern using multiple GPU threads for SpMV. In this figure, x is the projection vector, col_idx and w_val are the column index (in original SM) and value arrays of Sub_SM_3, and y is the resultant vector. Our task-mapping scheme is assigning one sub-matrix row to one thread, and each thread is demanded to compute the product of assigned row and x. This One-Thread-to-One-Row mapping can fully utilize the hardware resources, since the number of rows (e.g. 92160 in 720 views with 1024 rays per view case) is usually much larger than the number of commodity GPU cores (e.g. 4992 in Tesla K80). Moreover, in order to achieve coalesced memory access on GPU, we store the matrix in column major order. The SpMV computation can be executed following the statement in line 6 of Algorithm 2. For example, as shown in Fig. 10a, the thread 2(T2) handles the row 2 in the dense sub-matrix; T2 calculates the product of each row 2 element and its corresponding (based on value in col_idx) element of x, accumulates these products, and then adds the result to corresponding element of y (y_2 in this case).

When the kernel computes SpMV_T, it can leverage the same data layout used for SpMV. Figure 10b illustrates the data access pattern for SpMV_T: each thread still processes a compressed row; however, instead of accoplishing the accumulation of element-wise products within each thread, threads need to accumulate the product to corresponding elements of y', with the element indices determined by col_idx. For example, thread 1(T1) in Fig. 10b multiplies w_val_{10} by x_1 and accumulates the product to y'_2, then multiplies w_val_{11} by x_1 and accumulates the product to y'_4, and so on. Since we permute data inside each sub-matrix row to make the elements of a col_idx segment that for one sub-matrix column to have different values (e.g. elements of the first col_idx segment that for sub-matrix column 1 has the value “11”, “2”, “9”, “3”, “4”), different threads will always accumulate the products to different positions in y' at the same time point. Hence we can avoid the atomic operations.

Algorithm 2 SpMV and SpMV_T Kernel in cuART

```
1  Kernel MULTIPLICATION (w_val, col_idx, x, y, y')
2      $tid \longleftarrow$ blockIdx.x $*$ blockDim.x $+$ threadIdx.x;
3      /* SpMV */
4      $sum \longleftarrow 0$;
5      for $j \longleftarrow 0$ to $q$ do
6          $sum \longleftarrow sum + w\_val[j*p+tid] * x[col\_idx[j*p+tid]]$;
7      end
8      $y[tid] \longleftarrow sum$;
9      /* SpMV_T */
10     for $j \longleftarrow 0$ to $q$ do
11         $y'[col\_idx[j*p+tid]] \longleftarrow w\_val[j*p+tid] * x[tid]$;
12     end
```

6 Evaluation

We evaluate our cuART on the platform that has Intel Xeon E5-2697 multicore CPU running on 2.70GHz, 256GB system memory, and NVIDIA Tesla K80 GPU. The K80 is the newest model of Kepler architecture-based Tesla GPU, which has a total of 4992 CUDA cores and 24GB GDDR5 global memory. Our experimental data includes both standard test dataset – *Shepp-Logan phantom* [45] that serves as the human head model, and two real-world mouse datasets: mouse437 and mouse459. The cuART reconstructs images with 2048 $\times$ 2048, 1024 $\times$ 1024 and 512 $\times$ 512 resolutions respectively for each dataset. The parameters of the CT device are configured to 720 views and 1024 rays per view for each scan.

We first make a performance comparison between cuART and existing approaches. We take an existing single threaded CPU implementation [5] as baseline; it implements the original ART algorithm as shown in Eq. 1, which has a faster convergence speed but is not feasible to be parallelized. We compare cuART with GPU counterparts using CSR (cuSPARSE), BRC, and CSR5 respectively. As mentioned in the third row of Table 1, when using CSR format, we need 37.41 GB memory to hold such compressed system matrix for images at 2048 $\times$ 2048 resolution, which exceeds the global memory size of NVIDIA K80. BRC and CSR5 formats require even more memory spaces than CSR. As a result, in all GPU counterparts, we have to partition the system matrix into multiple groups and pipeline GPU computations and host-device data transfers, although this method leads to additional overhead. Furthermore, all counterparts use the $csr2csc$ function in cuSPARSE to explicitly transpose the system matrix for SpMV_T. These transpositions make the overhead larger since we have to also pipeline GPU computations and data transfers for the transposed matrix. All GPU versions implement the adapted ART algorithm in Eq. 9 that is easier to be parallelized but needs more iterations to reach convergence.

Figure 11 shows the execution time of reconstruction schemes, and Fig. 12 shows the performance comparisons. Both the CPU and GPU versions run adequate iterations to reach convergence. We find that only the image resolution affects reconstruction efficiency independent of the datasets; hence we don't distinguish the performances for different datasets. The figure shows CSR and BRC can achieve up to 3.7-fold speedup over CPU version, while CSR5 can achieve up to 4.8-fold speedup. Our cuART with SCSR and global-level blocking (g-l SCSR) can achieve up to 6.1, 6.6, 4.2-fold speedup against cuSPARSE, BRC and CSR5 respectively, while cuART with SCSR and view-level blocking (v-l SCSR) can achieve up to 6.8, 7.2, 5.4-fold speedup over aforementioned counterparts. BRC and CSR5 underperform their claimed best achievements because of

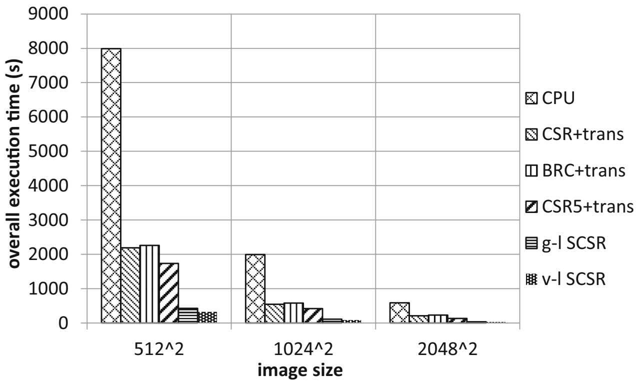


Figure 11 Execution time of ART-based CT image reconstructions on CPU and on GPU using CSR, BRC, CSR5, and SCSR respectively.

(1) the slower convergence speed than the CPU version, (2) the overhead of processing a mass of control tags and flags inside the data structures, (3) the time spent on the explicit transposition from CSR to CSC, and (4) the overhead in the pipeline mode to overlap GPU computations and host-device data transfers. On the other hand, SCSR can avoid or reduce all these overheads; hence g-l SCSR and v-l SCSR can acheive significantly improved performances compared to the GPU counterparts.

All GPU implementations use the parallelized ART that sacrificing the convergence speed for the parallelism. Among these GPU implementations, our cuART with v-l SCSR compensates the convergence speed loss to some extent due to the leveraging of finer sparsities hence converges faster than the others. Figure 13 is the comparison of Root Mean Square Error (RMSE) after each iteration between the original ART on CPU, the parallelized ART with g-l SCSR (same results with BRC or CSR5) and v-l SCSR respectively on GPU. It shows that original ART reaches convergence after running about 15 iterations, while parallelized ART needs 50 iterations when using g-l SCSR or 30 iterations when using v-l SCSR to converge. Since

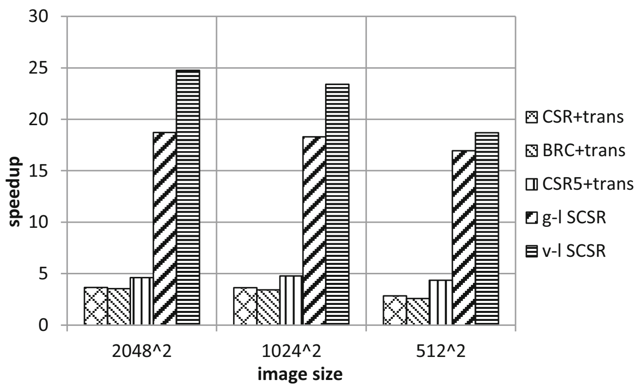


Figure 12 Performance comparisons between GPU versions of ART-based CT image reconstructions using CSR, BRC, CSR5, and SCSR respectively. The baseline is the single threaded CPU counterpart.

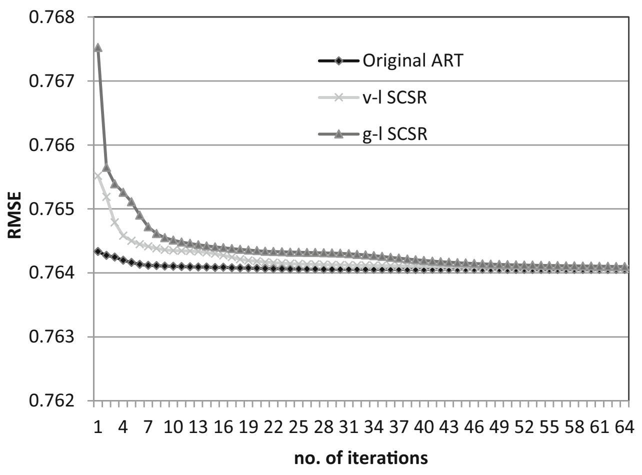


Figure 13 Convergence speed comparison between the original ART and the parallelized ART.

a single iteration on GPU is 62.4 times faster than one iteration on CPU, the GPU version with g-l SCSR can achieve 18.7-fold overall speedup over the CPU version. And, albeit one single GPU iteration with v-l SCSR is slightly slower than one iteration with g-l SCSR, v-l SCSR based GPU implementation can achieve 1.3-fold overall speedup against g-l SCSR based implementation and consequently 24.8-fold speedup against CPU counterparts, due to the 1.7 times faster convergence speed.

Both the SCSR data format and the parallelized ART do not affect the quality of reconstructed image.

Figure 14 shows the images reconstructed through (b) global-level (g-l) SCSR and (c) view-level (v-l) SCSR respectively, compared to (a) original phantom image. The differences among these images are already too subtle to be recognized by human eye perception. We then use three statistical measurements to quantitatively evaluate the reconstructed images' quality. We measure the images both globally and locally. Root Mean Square Error (RMSE) is one of the most commonly used metric for measuring global differences of the images. From Fig. 13 we can see, although g-l SCSR and v-l SCSR have different convergence speed, they finally achieve approximately the same RMSE compared to original ART after running adequate iterations. We further apply the structural similarity (SSIM) index method to verify the global evaluation. SSIM [46] is considered as the improvement of RMSE by eliminating the inconsistence between RMSE and human eye perception. Instead of estimating absolute errors by RMSE, SSIM adopts the perception-based model considering image degradation as perceived change in structural information. Table 2 shows the SSIM of the images to the reference. SSIM value is within the range from 0 to 1, and the closer to 1 means more similar to the reference. The results indicate our reconstructed images can be considered structurally the same as original phantom image. Finally, we locally measure the image qualities through line profiling. Line profile provides pixel-wise evaluations along an arbitrary line segment on the images (at the same relative positions on different images). Figure 15 shows the profiling results. The x-axis represents the indices of profile points along the line segment; the y-axis represents the average numerical value of the sample pixels around each profile point. From the figure we can see, except some slight noises at the edges, the reconstructed images are considered highly consistent to the original image. Based on above analyses, we can summarize that parallelized ART with our SCSR achieves the same reconstruction quality as the original ART.

Table 2 SSIM contrast using Phantom as reference.

	g-l SCSR	v-l SCSR	Phantom
SSIM	0.9878	0.9901	1

Figure 16 shows the memory footprint (dominated by system matrix) comparison between different compressed data formats. The memory footprint is dominated by the system matrix whose size is determined by image resolution and CT device configuration independent of the datasets. The g-l SCSR requires 7.8, 7.7 and 7.6 times less memory space than CSR for 2048^2, 1024^2 and 512^2

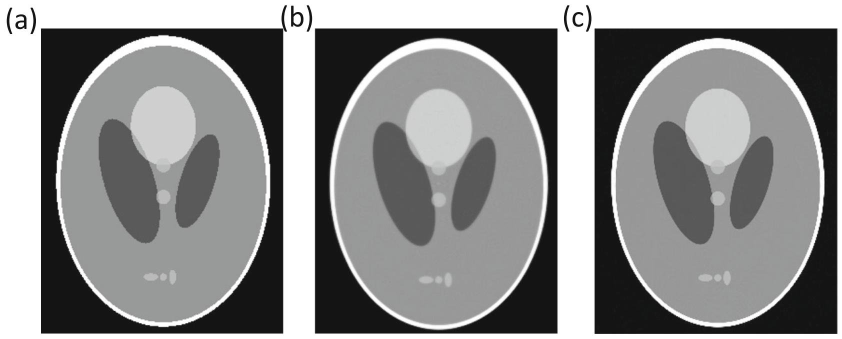


Figure 14 Reconstructed images using **b** g-l SCSR, **c** v-l SCSR respectively, compared to **a** original Phantom image. All images are in size 1024^2.

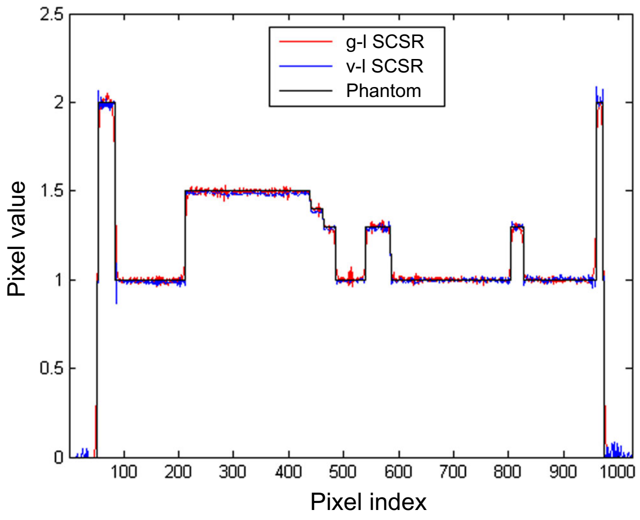


Figure 15 Line profiling on reconstructed images using g-l SCSR and v-l SCSR respectively, compared to original Phantom image.

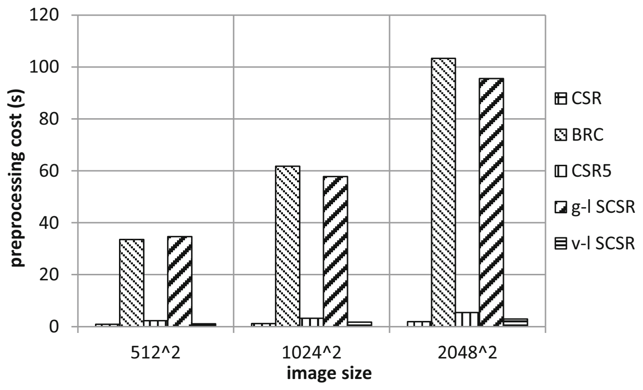


Figure 17 Preprocessing cost comparison. Vertical axis represents the preprocessing time in seconds.

image respectively, while the v-l SCSR requires 7.9, 7.9 and 7.8 times less memory space than CSR. BRC requires slightly larger memory space than CSR due to the zero paddings, and CSR5 requires nearly the same memory size as CSR since both two are padding-free. By leveraging the symmetry characteristics in the system matrix, SCSR doesn't store symmetric nonzero elements and needs only very few zero-paddings and a small bitmask, hence can achieve the best memory efficiency. Overall, SCSR is the only one data format that can make all system matrices in our experiments fit into the global memory of a single NVIDIA Tesla K80 GPU.

We also evaluate the preprocessing time to transform original system matrices to corresponding compressed data formats. Similar to the memory footprint, the preprocessing time positively correlates to image resolution independent of the datasets. Figure 17 is the preprocessing time comparison. It shows that CSR needs the least preprocessing time and can be treated as the baseline because all the other formats are its variants. It also shows BRC and g-l SCSR require substantial preprocessing time due to the sorting step. Preprocessing time of CSR5 is up to 2.9 times slower than CSR due to the dense sub matrices generation. Our v-l SCSR has nearly the same preprocessing cost as CSR because it doesn't spend much time on the dense sub matrices generation, thanks to the direct mapping by leveraging the sparsity pattern in each view.

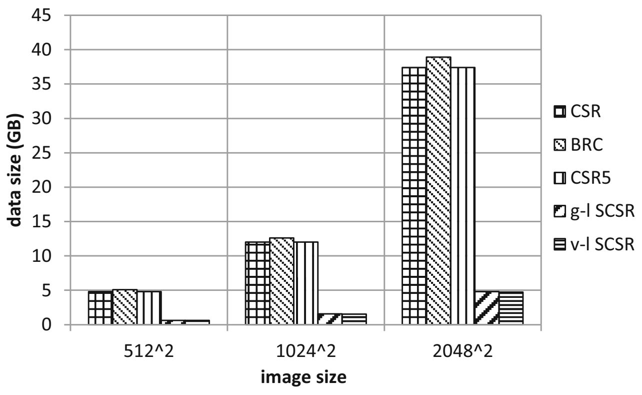


Figure 16 Memory footprint comparison in all schemes.

7 Conclusions

In this work, we propose cuART, an enhanced tool to accelerate ART-based computed tomography image reconstruction using GPUs. It consists of the SCSR data format and transpose-free GPU kernel for ART. SCSR is a variant of CSR that further compresses the CSR-based system matrix of CT scan by leveraging symmetry characteristics, and it optimizes data access for both SpMV and SpMV_T through column indices permutation. We also propose two blocking techniques to convert SCSR to several dense sub matrices by taking advantage of various sparsities of the system matrix; both blocking techniques can optimize workload distributions and computations. A transpose-free GPU kernel is designed to implement parallel ART algorithm by applying SCSR and blocking techniques to it. Our experiments illustrate cuART can achieve up to 6.8, 7.2 and 5.4-fold speedup over the GPU counterparts using cuSPARSE, BRC, and CSR5 on NVIDIA Tesla K80, respectively.

8 Discussions and Future Work

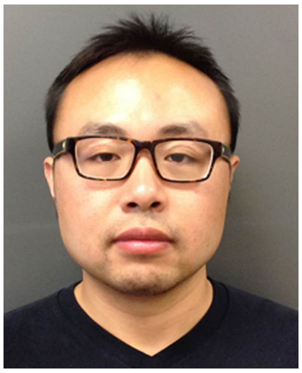
The Low-Dose CT following the well-known ALARA (as low as reasonably achievable) principle gradually becomes one of the popular research topics in the medical imaging domain [47]. This trend is led by the raising concerns on the side effect of radiations inducing genetic, cancerous and other diseases. In order to reduce the radiation exposures to the patients, one of the major methods in Low-Dose CT is reducing the scanning angles. Obviously, albeit we are not initially targeting the Low-Dose CT, our symmetry-based compression is naturally beneficial to the Low-Dose CT case since we reduce the required projection data for reconstructions and accordingly reduce the necessary scanning angles to one eighth of regular CT. On the other hand, we can consider to apply the state-of-the-art algorithms in Low-Dose CT such as total variation (TV) [48], Haar transform [49], and Shearlet-based regularization [50] to our cuART, to further reduce the data residing in device global memory.

In the future, we will also extend our cuART to reconstruct the 3D image. Iterative 2D and 3D image reconstruction algorithms essentially have the same basis, and hence our memory and computationally efficient approach may also significantly benefit the 3D cases. We also plan to implement cuART on large-scale GPU clusters by using modern parallel and distributed programming models, e.g. MPI, to satisfy the demands in the BIGDATA era.

References

1. IMV Medical Information Division (2007). IMV 2006 CT Market Summary Report Table of Contents.
2. Gordon, R., Bender, R., Herman, G. (1970). Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *Journal of Theoretical Biology*, *29*(3), 471–481.
3. Guan, H., & Gordon, R. (2005). A projection access order for speedy convergence of art (algebraic reconstruction technique): A multilevel scheme for computed tomography. *Physics in Medicine and Biology*, *39*(11), 1994.
4. Mueller, K., Yagel, R., Cornhill, J.F. (1997). The weighted-distance scheme: a globally optimizing projection ordering method for art. *IEEE Transactions on Medical Imaging*, *16*(2), 223–230.
5. Zhang, S., Zhang, D., Gong, H., Ghasemalizadeh, O., Wang, G., Cao, G. (2014). Fast and accurate computation of system matrix for area integral model-based algebraic reconstruction technique. *Optical Engineering*, *53*(11), 113101:1–113101:9.
6. Laurent, C., Peyrin, F., Chassery, J.-M., Amiel, M. (1998). Parallel image reconstruction on mimd computers for three-dimensional cone-beam tomography. *Parallel Computing*, *24*(9), 1461–1479.
7. Melvin, C. (2006). Design, Development and Implementation of a Parallel Algorithm for Computed Tomography Using Algebraic Reconstruction Technique. Canadian theses. University of Manitoba (Canada).
8. Grüll, F., Kunz, M., Hausmann, M., Kebschull, U. (2012). An implementation of 3d electron tomography on fpgas. In *2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig)* (pp. 1–5).
9. Pang, W.-M., Qin, J., Lu, Y., Xie, Y., Chui, C.-K., Heng, P.-A. (2011). Accelerating simultaneous algebraic reconstruction technique with motion compensation using cuda-enabled gpu. *International Journal of Computer-Assisted Radiology and Surgery*, *6*(2), 187–199.
10. Zhao, X., Hu, J.-J., Yang, T. (2013). Gpu-based iterative cone-beam ct reconstruction using empty space skipping. *Journal of X-ray Science and Technology*, *21*(1), 53–69.
11. Liu, R., Luo, Y., Yu, H. (2014). Gpu-based acceleration for interior tomography. *IEEE Access*, *2*, 757–770.
12. Guo, M., & Gao, H. (2017). Memory-efficient algorithm for stored projection and backprojection matrix in helical ct. *Medical Physics*, *44*(4), 1287–1300.
13. Ashari, A., Sedaghati, N., Eisenlohr, J., Sadayappan, P. (2014). An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on gpus. In *28th ACM Int'l Conf. on Supercomputing* (pp. 273–282).
14. Buluç, A., Fineman, J., Frigo, M., Gilbert, J., Leiserson, C. (2009). Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *21st ACM Symposium on Parallelism in Algorithms and Architectures* (pp. 233–244).
15. Liu, W., & Vinter, B. (2015). Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In *29th ACM Int'l Conf. on Supercomputing, ICS '15* (pp. 339–350).
16. Yu, X., Wang, H., Feng, W.-C., Gong, H., Cao, G. (2016). cuart: Fine-grained algebraic reconstruction technique for computed tomography images on gpus. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (pp. 165–168).
17. Yu, X., Wang, H., Feng, W.-C., Gong, H., Cao, G. (2017). An enhanced image reconstruction tool for computed tomography on gpus. In *Proceedings of the Computing Frontiers Conference, CF'17* (pp. 97–106): ACM.
18. Kak, A.C. (1984). Image Reconstruction from Projections. In Ekstrom, M. (Ed.) *Digital Image Processing Techniques, chapter 4,* (pp. 111–171). Orlando: Academic Press, INC.
19. Gilbert, P. (1972). Iterative methods for the three-dimensional reconstruction of an object from projections. *Journal of theoretical biology*, *36*(1), 105–117.
20. Andersen, A.H., & Kak, A.C. (1984). Simultaneous algebraic reconstruction technique (sart): a superior implementation of the art algorithm. *Ultrasonic Imaging*, *6*(1), 81–94.
21. Liu, W., & Vinter, B. (2015). Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Computing*, *49*, 179–193.
22. Yan, S., Li, C., Zhang, Y., Zhou, H. (2014). yaspmv: Yet another spmv framework on gpus. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '14* (pp. 107–118): ACM.
23. Ashari, A., Sedaghati, N., Eisenlohr, J., Parthasarath, S., Sadayappan, P. (2014). Fast sparse matrix-vector multiplication on gpus for graph applications. In *SC14* (pp. 781–792).
24. Greathouse, J., & Daga, M. (2014). Efficient sparse matrix-vector multiplication on gpus using the csr storage format. In *SC14* (pp. 769–780).
25. Merrill, D., & Garland, M. (2016). Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format. In *21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '16* (pp. 43:1–43:2).
26. Steinberger, M., Zayer, R., Seidel, H.-P. (2017). Globally homogeneous, locally adaptive sparse matrix-vector multiplication on the gpu. In *Proceedings of the International Conference on Supercomputing, ICS '17* (pp. 13:1–13:11). New York: ACM.
27. Hou, K., Feng, W.-C., Che, S. (2017). Auto-tuning strategies for parallelizing sparse matrix-vector (spmv) multiplication on multi- and many-core processors. In *2017 IEEE International Parallel*

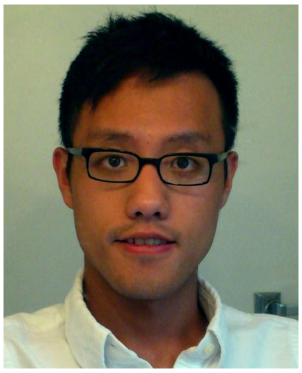
and Distributed Processing Symposium Workshops (IPDPSW) (pp. 713–722).
28. Wang, H., Liu, W., Hou, K., Feng, W.-C. (2016). Parallel transposition of sparse data structures. In *Proceedings of the International Conference on Supercomputing, ICS '16* (p. 2016).
29. Nourian, M., Wang, X., Yu, X., Feng, W.-C., Becchi, M. (2017). Demystifying automata processing: Gpus, fpgas or micron's ap? In *Proceedings of the International Conference on Supercomputing, ICS '17* (pp. 1:1–1:11). New York: ACM.
30. Hou, K., Wang, H., Feng, W.-C. (2015). *Aspas: A framework for automatic simdization of parallel sorting on x86-based many-core processors*, (pp. 383–392). New York: ACM.
31. Yu, X., Hou, K., Wang, H., Feng, W.-C. (2017). A framework for fast and fair evaluation of automata processing hardware. In *2017 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 120–121).
32. Yu, X., Hou, K., Wang, H., Feng, W.-C. (2017). Robotomata: A framework for approximate pattern matching of big data on an automata processor. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 283–292).
33. Yu, X., Lin, B., Becchi, M. (2014). Revisiting state blow-up: Automatically building augmented-fa while preserving functional equivalence. *IEEE Journal on Selected Areas in Communications*, *32*(10), 1822–1833.
34. Yu, X., Feng, W.-C., Yao, D., Becchi, M. (2016). O3fa: A scalable finite automata-based pattern-matching engine for out-of-order deep packet inspection. In *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (pp. 1–11).
35. Yu, X., & Becchi, M. (2013). Gpu acceleration of regular expression matching for large datasets: Exploring the implementation space. In *ACM Int'l Conf. on Computing Frontiers, CF '13* (pp. 18:1–18:10). New York: ACM.
36. Zhang, J., Wang, H., Feng, W.-C. (2015). cublastp: Fine-grained parallelization of protein sequence search on cpu+gpu. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *PP*(99), 1–1.
37. Yu, X., & Becchi, M. (2013). Exploring different automata representations for efficient regular expression matching on gpus. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13* (pp. 287–288). New York: ACM.
38. Hou, K., Liu, W., Wang, H., Feng, W.-C. (2017). Fast segmented sort on gpus. In *Proceedings of the International Conference on Supercomputing, ICS '17* (pp. 12:1–12:10). New York: ACM.
39. Yu, X. (2013). Deep packet inspection on large datasets: Algorithmic and parallelization techniques for accelerating regular expression matching on many-core processors. Master's thesis, University of Missouri–Columbia.
40. Keck, B., Hofmann, H., Scherl, H., Kowarschik, M., Hornegger, J. (2009). Gpu-accelerated sart reconstruction using the cuda programming environment. In *SPIE Medical Imaging* (pp. 72582B–72582B): International Society for Optics and Photonics.
41. Naumov, M., Chien, L.S., Vandermersch, P., Kapasi, U. (2010). cusparse library. In *GPU Technology Conference*.
42. Aktulga, H.M., Buluç, A., Williams, S., Yang, C. (2014). Optimizing sparse matrix-multiple vectors multiplication for nuclear configuration interaction calculations. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium* (pp. 1213–1222).
43. Nickolls, J., Buck, I., Garland, M., Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, *6*(2), 40–53.
44. Tao, Y., Deng, Y., Mu, S., Zhang, Z., Zhu, M., Xiao, L., Ruan, L. (2015). Gpu accelerated sparse matrix-vector multiplication and sparse matrix-transpose vector multiplication. *Concurrency and Computation: Practice and Experience*, *27*(14), 3771–3789.
45. Shepp, L.A., & Logan, B.F. (1974). The fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science*, *21*(3), 21–43.
46. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612.
47. Xu, Q., Yu, H., Mou, X., Zhang, L., Hsieh, J., Wang, G. (2012). Low-dose x-ray ct reconstruction via dictionary learning. *IEEE Transactions on Medical Imaging*, *31*(9), 1682–1697.
48. Du, Y., Wang, X., Xiang, X., Wei, Z. (2016). Evaluation of hybrid SART+OS+TV iterative reconstruction algorithm for optical-CT gel dosimeter imaging. *Physics in Medicine & Biology*, *61*(24), 8425.
49. Garduño, E., Herman, G.T., Davidi, R. (2011). Reconstruction from a few projections by 1 -minimization of the Haar transform. *Inverse Problems*, *27*(5), 055006.
50. Vandeghinste, B., Goossens, B., Van Holen, R., Vanhove, C., Piurica, A., Vandenberghe, S., Staelens, S. (2013). Iterative ct reconstruction using shearlet-based regularization. *IEEE Transactions on Nuclear Science*, *60*(5), 3305–3317.



Xiaodong Yu is currently a Ph.D. student in the Department of Computer Sciences at Virginia Tech. He received the B.S. degree in mathematics from China University of Mining and Technology (CUMT), Xuzhou, China, and the M.S. degree in electrical and computer engineering from University of Missouri, Columbia, MO. His research interests are primarily in High-Performance Computing, with a focus on architecture- and application- aware algorithms optimization and acceleration.

Hao Wang is a Senior Research Associate in the Dept. of Computer Science at Virginia Tech. He received his Ph.D. in the Institute of Computing Technology at Chinese Academy of Sciences. His research interests include high performance, parallel, and distributed computing. His current focus is on designing algorithms of graph processing, bioinformatics, and computational fluid dynamics on HPC clusters with InfiniBand interconnects, GPUs and Intel MIC co-processors.

Wu-chun Feng is a professor and Elizabeth & James E. Turner Fellow in the Dept. of Comp. Science, Dept. of Electrical and Computer Engineering, Health Sciences, and Virginia Bioinformatics Institute at Virginia Tech. His interests lie broadly at the synergistic intersection of computer architecture, systems software and applications software. Most recently, his research has dealt with accelerator-based computing for bioinformatics and power-aware computing.



Hao Gong is currently a research fellow in Department of Radiology, Mayo Clinic, MN, USA. He received his PhD degree from School of Biomedical Engineering and Sciences, Virginia Polytechnic Institute and State University, VA, USA, in 2017. His research is within the field of Clinical X-ray Computed Tomography, and machine learning for Clinical applications.

Guohua Cao is an Assistant Professor of Biomedical Engineering in the Department of Biomedical Engineering and Mechanics at Virginia Tech in Blacksburg, VA. He received his PhD in Physical Chemistry from Brown University in 2005. After postdoctoral trainings at Brown University and UNC-Chapel Hill from 2005 to 2007, he became a Research Assistant Professor in the Department of Physics and Astronomy at UNC-Chapel Hill from 2008 to 2011. Dr. Cao has been recognized by a number of awards including a NSF CAREER award in 2014. Dr. Cao's research is directed at biomedical imaging, with a focus on developing novel imaging hardware and software for image acquisition and formation.