An Enhanced Image Reconstruction Tool for Computed Tomography on GPUs

Xiaodong Yu, Hao Wang, Wu-chun Feng
Dept. of Computer Science, Virginia Tech
Blacksburg, VA 24060, USA
{xdyu,hwang121,wfeng}@vt.edu

Hao Gong, Guohua Cao
Dept. of Biomedical Engr. and Mechanics, Virginia Tech
Blacksburg, VA 24060, USA
{haog1,ghcao}@vt.edu

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 enhanced 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 and sorting-free blocking techniques to optimize the kernel computation by leveraging the 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.

CCS CONCEPTS

•Computing methodologies → Linear algebra algorithms; Parallel algorithms; •Applied computing → Life and medical sciences;

KEYWORDS

GPU, Computed Tomography, Image Reconstruction, Algebraic Reconstruction Technique, Sparse Matrix-Vector Multiplication, SpMV, Transposition

CF'17, Siena, Italy

DOI: http://dx.doi.org/10.1145/3075564.3078889

ACM Reference format:
Xiaodong Yu, Hao Wang, Wu-chun Feng and Hao Gong, Guohua Cao. 2017. An Enhanced Image Reconstruction Tool for Computed Tomography on GPUs. In *Proceedings of CF'17, Siena, Italy, May 15-17, 2017,* 10 pages.
DOI: http://dx.doi.org/10.1145/3075564.3078889

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 [13]. 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) [7]. 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 [10, 21, 38], while others try to map ART to high-performance computing (HPC) platforms, e.g., multicore processors [15], Beowulf clusters [19], and FPGAs [9]. Recently, GPUs have been employed as viable accelerators for ART-based CT image reconstruction [11, 16, 24, 39], 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 [16]) 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 [3, 4, 17].

In this paper, we propose our own enhanced CUDA-accelerated algebraic reconstruction technique (i.e., cuART), which provides a complete compression and parallelization solution for ART-based CT image reconstruction on GPUs. Our preliminary work [33] briefly introduced the basic idea of our symmetry-based compression and sorting-based format transformation. In this paper, we provide detailed methodologies and designs of our symmetry-based, CSR format (*SCSR*) and a global, sorting-based, blocking technique as well as comprehensive analyses of their bases, i.e., the symmetries and global-level sparsity pattern of the system matrix. Moreover,

we propose three new designs for this enhanced cuART that are not found in [33].

First, we 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). Second, beyond the sorting-based blocking scheme in [33], we propose a sorting-free blocking mechanism to transform the SCSR format into multiple dense sub-matrices at runtime by leveraging the view-level sparsity patterns of the system matrix and aiming to reduce the preprocessing overhead. Third, 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 to achieve significant performance improvements over other general SpMV schemes on GPUs. Our major contributions can be summarized as follows:

- 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.
- A sorting-free blocking technique to convert SCSR to dense sub-matrices at runtime by leveraging view-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: $\boldsymbol{WF=P}$, 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 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.

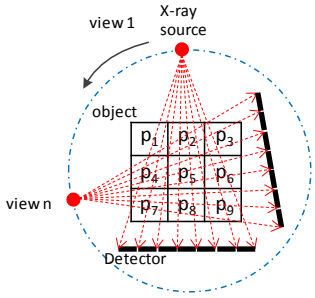

Figure 1: Schematic diagram of the spiral CT structure.

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 [14]. So, Gordon et al. [7] 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:

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

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 Equation (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) [6] and (2) simultaneous algebraic reconstruction technique (SART) [1]. 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.

[1] Compared to the approaches in [33], our view-level, sparsity-based, sorting-free blocking mechanism 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.

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 [24, 39]. However, these approaches introduce large computational overhead due to repeated on-the-fly computations during iterative image reconstruction [11]. 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) [38] 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 Figure 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 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 Equation (1) falls into category of sparse matrix-vector multiplication (SpMV).

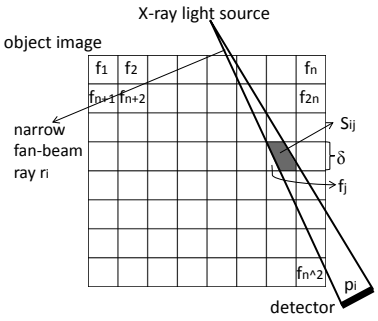

Figure 2: Fan-beam area integral model of CT.

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 5(a)-(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) [3] 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 [17] evenly partitions non-zero elements into multiple 2D tiles to achieve load balance and avoid storing unnecessary data; and its extension [18] directly uses the CSR format for load balanced SpMV and also considers CPU-GPU heterogeneous processors. Compressed sparse blocks (CSB) [4] 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 [29] provides an efficient segmented sum algorithm for the blocked compressed common coordinate (BCCOO) format with an auto-tuning framework for parameter selection. ACSR [2] leverages binning and dynamic parallelism techniques provided by modern GPUs to reduce thread divergence for the standard CSR. CSR-Adaptive [8] uses local scratchpad memory and dynamically assigns various number of rows to GPU compute units. More recently, NVIDIA proposed a merge-based parallel SpMV [20], which uses CSR to minimize preprocessing overhead and applies equitable multi-partitioning on the inputs to ensure load balance. 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 [27]. Thus, transpose-free methods for SpMV and SpMV_T are in urgent need.

2.4 GPU-based CT Image Reconstruction

GPUs continue to be used to accelerate a myriad of applications [5, 12, 28, 30–32, 34–37]. For CT image reconstruction, many researchers have also looked into how to leverage GPUs. For example, Pang et al. [24] propose a ray-voxel hybrid driven method to map SART onto GPU architecture, while Zhao et al. [39] 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. [11] 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. [16] accelerate SART with cuSPARSE [22] 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. In the remaining sections of this paper, we utilize the NVIDIA GPU architecture and its Compute Unified Device Architecture (CUDA) [23] 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. 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.

Table 1: Sizes of CSR-based compressed system matrices

projection_view#	360			720		
image_size(pixel#)	512^2	1024^2	2048^2	512^2	1024^2	2048^2
matrix_size (GB)	2.34	6.19	18.44	4.82	11.97	37.41

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

Our preliminary work [33] introduced the basic idea of symmetry-based compression in addition to the data structure-level compression e.g. CSR. In this subsection, we elaborate on the physical symmetry characteristics of CT scan and provide a detailed methodology for reducing the number of elements that must be stored for our SCSR format.

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.

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 Figure 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 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:

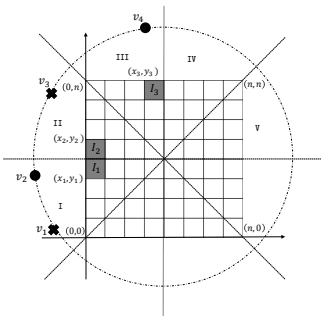

Figure 3: Schematic diagram of views/X-rays symmetries.

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

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 Equation (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. Table 2 shows the data sizes of some SCSR-based system matrices. Compared to data sizes in Table 1, SCSR can fit into the global memory of a single GPU even in our largest case.

Table 2: Sizes of SCSR-based compressed system matrices

projection_view#	360			720		
image_size(pixel#)	512^2	1024^2	2048^2	512^2	1024^2	2048^2
matrix_size (GB)	0.29	0.77	2.31	0.60	1.50	4.68

3.2 Column Indices Based Permutation for SCSR

As shown in Equation (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, thisincreases 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 [4], 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 Equation (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 [26].

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 Figure 5(c), the last two elements in the second column with "4" (i.e., "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 Figure 5(d) 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 [6], we know that processing the whole system matrix simultaneously is feasible. [33] introduced the preliminary idea of a sorting-based blocking. In this subsection, we provide the methodology of leveraging the global-level sparsity pattern of system matrix and detailed design of the blocking technique.

Figure 4 shows the methodology. After shifting all nonzero elements to the left, the system matrix 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

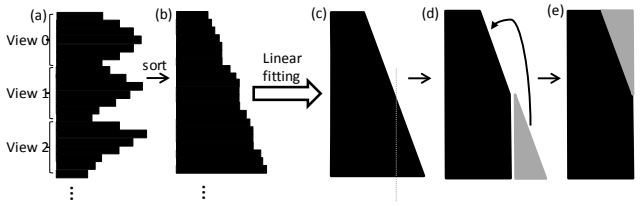

Figure 4: Schematic diagram for the methodology of leveraging the *global sparsity pattern* to block the SCSR data.

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 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 detailed design of the *global-level blocking* technique to transform CSR data to dense sub-matrices, as illustrated in Figure 5. Figure 5(a) and (b) are the layouts of system matrix in original representation and CSR format respectively. Figure 5(c) 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 Figure 5(d). 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 Figure 5(f): 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 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 works better than existing general blocking techniques in terms of CT image reconstruction performance, it remains two drawbacks: *a*) the heavy preprocessing overhead due to the sorting step which may become a performance

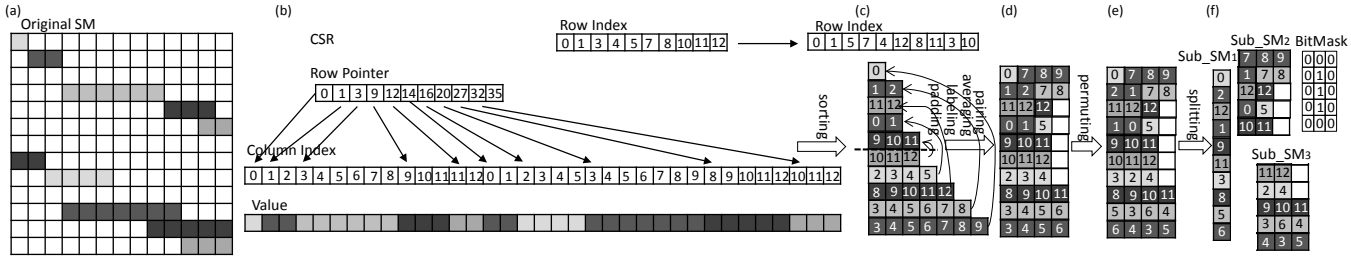

Figure 5: Schematic diagram of CSR to dense matrix based on global sparsity pattern of CT system matrix. White elements are zero paddings, numbers are column indices.

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, and 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 [1] theoretically approves 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. We introduce this scheme starting 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 of X-rays and the object image at each view position. Two inflection points on each shape are determined by two separator rays: one is the right-most ray and the other is the one going through lower-right image vertex.

We then provide the mathematical analyses for two non-visualized characteristics: *a*) positions of inflection points on shapes from adjacent views have constant distance, and *b*) view 5 is the border view that its left area of sparsity pattern has the valley slope (refer to characteristic (c) in last paragraph).

Figure 7 shows a sample view V, in which x is the light source, y is the center of both the image and virtual circle 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. Exploring how

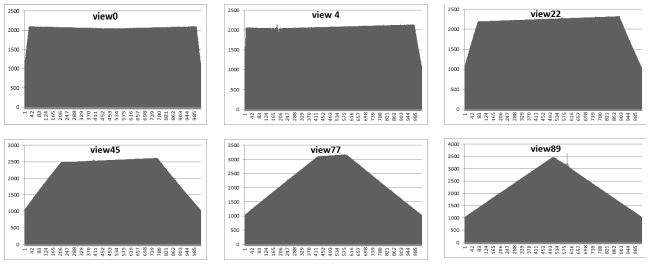

Figure 6: Sparsities of six typical projection views.

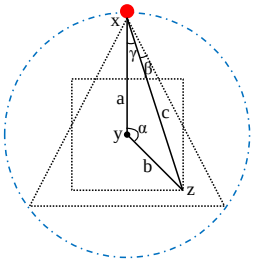

Figure 7: Schematic diagram of a sample view V.

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: $c^2 = a^2 + b^2 - 2ab\cos\alpha$ and $b^2 = a^2 + c^2 - 2ac\cos\gamma$. After applying substitutions to them, we have

$$\gamma = \arccos\left(\frac{a - b\cos\alpha}{\sqrt{a^2 + b^2 - 2ab\cos\alpha}}\right) \tag{3}$$

According to the differential formula $\mathrm{d}(\arccos x) = \left(1/\sqrt{1-x^2}\right)\mathrm{d}x$, if $\angle\gamma'$ is angularly similar enough to $\angle\gamma$, there is a equation:

$$\frac{\arccos\gamma - \arccos\gamma'}{\gamma - \gamma'} \approx \frac{1}{\sqrt{1-\gamma^2}} \tag{4}$$

Assuming $\angle\gamma'$ is the angle at the same position in view V' which is adjacent to V , we substitute Equation (3) into (4) and get:

$$\gamma - \gamma' = \frac{\frac{a-b\cos\alpha}{\sqrt{a^2+b^2-2ab\cos\alpha}} - \frac{a-b\cos\alpha'}{\sqrt{a^2+b^2-2ab\cos\alpha'}}}{\sqrt{1-\left(\frac{a-b\cos\alpha}{\sqrt{a^2+b^2-2ab\cos\alpha}}\right)^2}} \tag{5}$$

In practice, the X-ray light source moves with 0.5° angular step i.e. $\angle\alpha' = \angle\alpha + 0.5°$; hence the values of $\cos\alpha'$ and $\cos\alpha$ are close enough. a and b are fixed number once CT device is set. Accordingly, we approximately simplify Equation (5) and get:

$$\Delta\gamma \approx \frac{(\cos\alpha' - \cos\alpha)}{\sqrt{1-\cos^2\alpha}} \approx \cot\alpha' - \cot\alpha \tag{6}$$

Equation (6) 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.

Based on all above analyses, we propose the *view-level blocking* that transforms SCSR to dense sub-matrices at runtime. Figure 8 shows the schematic diagram for the view-level sparsities based transformation. 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. (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. According to the mathematical proof in the last paragraph, inflection point positions change regularly, hence 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 (view*a*, view(*90-a-1*), where *a* is an integer less than 45) by moving elements and add flags. (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.

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). 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 Equation (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

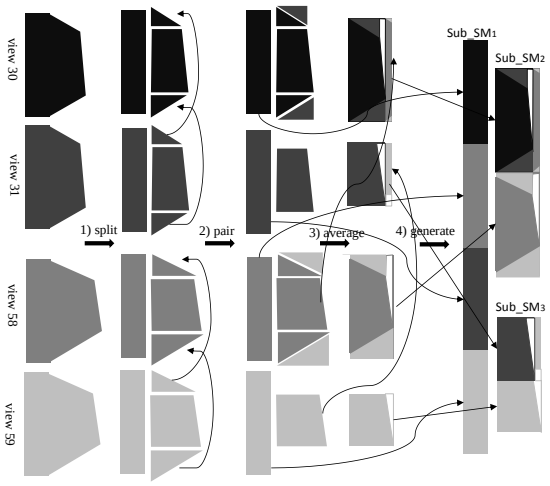

Figure 8: Schematic diagram of sample view for mathematical analysis. White areas are zero paddings.

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:

$$F^{(r+1)} = F^{(r)} + W^T(C. * (P - WF^{(r)})) \tag{7}$$

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 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.

Guan and Gordon [10] prove that changing the access order of X-rays and projection data 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, Equation (7) can achieve the same reconstructed image quality to Equation (1) after running adequate iterations. Although Equation (7) has slower convergence speed than Equation (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. Algorithm 1 shows such kernel.

Figure 9 (a) 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,

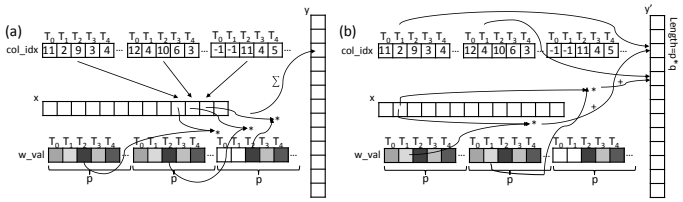

Figure 9: Matrix memory layout and (a) SpMV data access pattern and (b) SpMV_T data access pattern.

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 1. For example, as shown in Figure 9 (a), 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 9 (b) 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' addressed using *col_idx*. For example, thread 1(T1) in Figure 9(b) 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 1: SpMV and SpMV_T Kernel in cuART

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

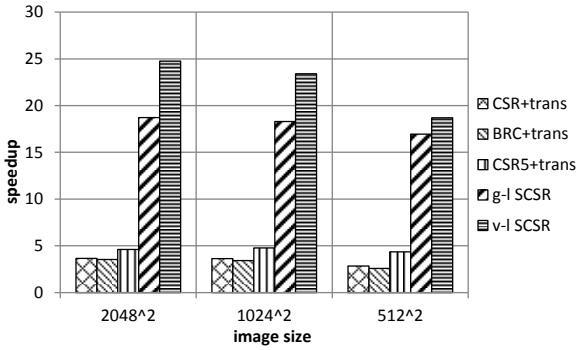

Figure 10: Performance comparisons between GPU versions of ART-based CT image reconstructions using CSR, BRC, CSR5, and SCSR respectively. The baseline is the single threaded CPU counterpart.

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* [25] that serves as the human head model, and two real-world mouse datasets: mouse437 and mouse459. The cuART reconstructs images with 2048×2048, 1024×1024 and 512×512 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 [38] as baseline; it implements the original ART algorithm as shown in Equation 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 Table 1, when using CSR format, we need 37.41 GB memory to hold such compressed system matrix for images at 2048×2048 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 Equation 7 that is easier to be parallelized but needs more iterations to reach convergence.

Figure 10 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

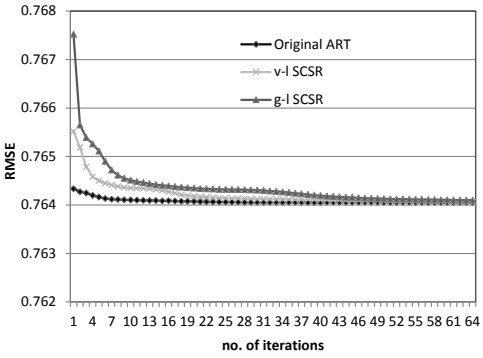

Figure 11: Convergence speed comparison between the original ART and the parallelized ART.

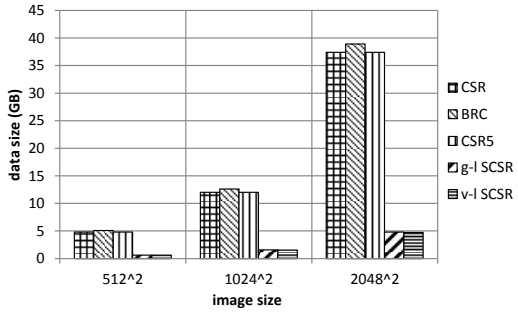

Figure 12: Memory footprint comparison in all schemes.

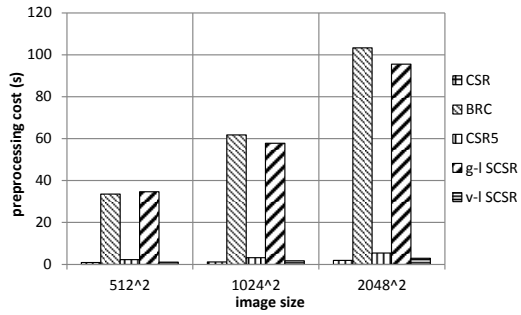

Figure 13: Preprocessing cost comparison. Vertical axis represents the preprocessing time in seconds.

up to 6.8, 7.2, 5.4-fold speedup over aforementioned counterparts. BRC and CSR5 underperform their claimed best achievements because of (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 11 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 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. Although a 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 one 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 11 shows, although g-l SCSR and v-l SCSR require 50 and 30 iterations while the original ART requires 15 iterations to reach convergence, the differences of convergent RMSEs between the original ART and parallelized ART with either g-l SCSR or v-l SCSR are negligible. Hence we can claim that parallelized ART with our SCSR achieves the same reconstruction quality as the original ART.

Figure 12 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 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 13 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.

7 CONCLUSIONS AND FUTURE WORK

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.

In the future, we will 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.

8 ACKNOWLEDGMENT

This work was supported in part by the NSF XPS program via CCF-1337131 and the Institute for Critical Technology and Applied Science (ICTAS), an institute dedicated to transformative, interdisciplinary research for a sustainable future. The authors also acknowledge Advanced Research Computing at Virginia Tech for providing computational resources.

REFERENCES

[1] A.H. Andersen and A.C. Kak. 1984. Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the ART Algorithm. *Ultrasonic Imaging* 6, 1 (1984), 81 – 94.

[2] A. Ashari, N. Sedaghati, J. Eisenlohr, S. Parthasarath, and P. Sadayappan. 2014. Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications. In *SC14*. 781–792.

[3] A. Ashari, N. Sedaghati, J. Eisenlohr, and P Sadayappan. 2014. An Efficient Two-Dimensional Blocking Strategy for Sparse Matrix-Vector Multiplication on GPUs. In *28th ACM Int'l Conf. on Supercomputing*. 273–282.

[4] A. Buluç, J. T Fineman, M. Frigo, J. R Gilbert, and C. E Leiserson. 2009. Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication using Compressed Sparse Blocks. In *21st ACM Symposium on Parallelism in Algorithms and Architectures*. 233–244.

[5] X. Cui, T. R. W. Scogland, B. R de Supinski, and W. Feng. 2016. Directive-Based Pipelining Extension for OpenMP. In *IEEE Cluster Computing*. 481–484.

[6] P. Gilbert. 1972. Iterative Methods for the Three-Dimensional Reconstruction of an Object from Projections. *Journal of Theoretical Biology* 36, 1 (1972).

[7] R. Gordon, R. Bender, and G. T Herman. 1970. Algebraic Reconstruction Techniques (ART) for Three-Dimensional Electron Microscopy and X-ray Photography. *Journal of Theoretical Biology* 29, 3 (1970), 471–481.

[8] J. L Greathouse and M. Daga. 2014. Efficient Sparse Matrix-Vector Multiplication on GPUs using the CSR Storage Format. In *SC14*. 769–780.

[9] F. Grüll, M. Kunz, M. Hausmann, and U. Kebschull. 2012. An implementation of 3D Electron Tomography on FPGAs. In *2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 1–5.

[10] H. Guan and R. Gordon. 1994. 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), 2005.

[11] M. Guo and H. Gao. 2017. Memory-Efficient Algorithm for Stored Projection and Backprojection Matrix in Helical CT. *Medical Physics* (2017).

[12] K. Hou, H. Wang, and W. Feng. 2017. GPU-UniCache: Automatic Code Generation of Spatial Blocking for Stencils on GPUs. In *ACM Int'l Conf. on Computing Frontiers (CF)*.

[13] IMV Medical Information Division. 2007. IMV 2006 CT Market Summary Report Table of Contents. (2007). http://www.imvinfo.com/user/documents/content_documents/nws_rad/MS_CT_DSandTOC.pdf

[14] A. C Kak. 1984. Image Reconstruction from Projections. *Digital Image Processing Techniques* (1984), 111–171.

[15] C. Laurent, F. Peyrin, J.-M. Chassery, and M. Amiel. 1998. Parallel Image Reconstruction on MIMD Computers for Three-Dimensional Cone-Beam Tomography. *Parallel Comput.* 24, 9 (1998), 1461–1479.

[16] R. Liu, Y. Luo, and H. Yu. 2014. GPU-Based Acceleration for Interior Tomography. *Access, IEEE* 2 (2014), 757–770.

[17] W. Liu and B. Vinter. 2015. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. In *29th ACM Int'l Conf. on Supercomputing (ICS '15)*. 339–350.

[18] W. Liu and B. Vinter. 2015. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Comput.* 49 (2015), 179–193.

[19] C. Melvin. 2006. *Design, Development and Implementation of a Parallel Algorithm for Computed Tomography Using Algebraic Reconstruction Technique*. University of Manitoba (Canada). https://books.google.com/books?id=Hih7hwkoFY4C

[20] D. Merrill and M. Garland. 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)*. Article 43, 2 pages.

[21] K. Mueller, R. Yagel, and J F. Cornhill. 1997. The Weighted-Distance Scheme: A Globally Optimizing Projection Ordering Method for ART. *IEEE Transactions on Medical Imaging* 16, 2 (1997), 223–230.

[22] M. Naumov, L.S. Chien, P. Vandermersch, and U. Kapasi. 2010. cuSPARSE Library. In *GPU Technology Conference*.

[23] J. Nickolls, I. Buck, M. Garland, and K. Skadron. 2008. Scalable Parallel Programming with CUDA. *Queue* 6, 2 (March 2008), 40–53.

[24] W.-M. Pang, J. Qin, Y. Lu, Y. Xie, C.-K. Chui, and P.-A. Heng. 2011. Accelerating Simultaneous Algebraic Reconstruction Technique with Motion Compensation using CUDA-enabled GPU. *International Journal of Computer-Assisted Radiology and Surgery* 6, 2 (2011), 187–199.

[25] L.A. Shepp and B.F. Logan. 1974. The Fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science* 21, 3 (June 1974), 21–43.

[26] Y. Tao, Y. Deng, S. Mu, Z. Zhang, M. Zhu, L. Xiao, and L. Ruan. 2014. GPU-Accelerated Sparse Matrix-Vector Multiplication and Sparse Matrix-Transpose Vector Multiplication. *Concurrency and Comp.: Practice and Experience* (2014).

[27] H. Wang, W. Liu, K. Hou, and W. Feng. 2016. Parallel Transposition of Sparse Data Structures. In *30th ACM Int'l Conf. on Supercomputing*. ACM, 33.

[28] H. Wu, D. Li, and M. Becchi. 2016. Compiler-Assisted Workload Consolidation for Efficient Dynamic Parallelism on GPU. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 534–543.

[29] S. Yan, C. Li, Y. Zhang, and H. Zhou. 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)*. ACM, 107–118.

[30] X. Yu. 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.

[31] X. Yu and M. Becchi. 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)*. ACM, New York, NY, USA, 287–288.

[32] X. Yu and M. Becchi. 2013. GPU Acceleration of Regular Expression Matching for Large Datasets: Exploring the Implementation Space. In *ACM Int'l Conf. on Computing Frontiers (CF '13)*. ACM, New York, NY, USA, Article 18, 10 pages.

[33] X. Yu, H. Wang, W. Feng, H. Gong, and G. Cao. 2016. cuART: Fine-Grained Algebraic Reconstruction Technique for Computed Tomography Images on GPUs. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 165–168.

[34] Y. Yuan, R. Lee, and X. Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proceedings of the VLDB Endowment* 6, 10 (2013), 817–828.

[35] Y. Yuan, F. Salmi, Y. Huai, K. Wang, R. Lee, and X. Zhang. 2016. Spark-GPU: An Accelerated In-Memory Data Processing Engine on Clusters. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 273–283.

[36] J. Zhang, H. Wang, and W.-c. Feng. 2015. cuBLASTP: Fine-Grained Parallelization of Protein Sequence Search on CPU+GPU. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* PP, 99 (2015), 1–1.

[37] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang. 2015. Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1226–1237.

[38] S. Zhang, D. Zhang, H. Gong, O. Ghasemalizadeh, G. Wang, and G. Cao. 2014. Fast and Accurate Computation of System Matrix for Area Integral Model-Based Algebraic Reconstruction Technique. *Optical Engineering* 53, 11 (2014).

[39] X. Zhao, J.-J. Hu, and T. Yang. 2013. GPU-Based Iterative Cone-Beam CT Reconstruction using Empty Space Skipping. *Journal of X-ray Science and Technology* 21, 1 (2013), 53–69.