cuART: Fine-Grained Algebraic Reconstruction Technique for Computed Tomography Images on GPUs

Xiaodong Yu, Hao Wang, Wu-chun Feng
Dept. of Computer Science, Virginia Tech
{xdyu,hwang121,feng}@cs.vt.edu

Hao Gong, Guohua Cao
Dept. of Biomedical Engineering and Mechanics, Virginia Tech
{haog1,ghcao}@vt.edu

Abstract—**Algebraic reconstruction technique (ART) is an iterative algorithm for computed tomography (CT) image reconstruction. Due to the high computational cost, researchers turn to modern HPC systems with GPUs to accelerate the ART algorithm. However, the existing proposals suffer from inefficient designs of compressed data structure and computational kernel on GPUs. In this paper, we identify the computational patterns in the ART as the product of a sparse matrix (and its transpose) with multiple vectors (SpMV and SpMV_T). Because the implementations with well-tuned libraries, including cuSPARSE, BRC, and CSR5, underperform the expectations, we propose cuART, a complete compression and parallelization solution for the ART-based CT on GPUs. Based on the physical characteristics, i.e., the symmetries in the system matrix, we propose the symmetry-based CSR format (SCSR), which can further compress data storage by removing symmetric but redundant non-zero elements. Leveraging the sparsity patterns of X-ray projection, we transform the CSR format to multiple dense sub-matrices in SCSR. We then design a transposition-free kernel to optimize the data access for both SpMV and SpMV_T. The experimental results illustrate that our mechanism can reduce memory usage significantly and make practical datasets fit into a single GPU. Our results also illustrate the superior performance of cuART compared to the existing methods on CPU and GPU.**

I. Introduction and motivation

Today, two categories of CT image reconstruction techniques are widely used. The algebraic reconstruction technique (ART) [1] is an iterative method that can provide better image quality with fewer projection views and less radiation dose than the analytical method, i.e., filtered back projection (FBP). Due to the high computational cost, researchers have invested significant effort in accelerating ART at both the algorithmic and runtime system level. The authors of [2], [3] work to speed up ART by modifying the algorithm while others map ART to HPC platforms having multi-core processors [4] and FPGAs [5]. Recently, GPUs are employed as a viable platform for ART-based CT applications [6], [7]. These GPU-based proposals have reported 10 to 15-fold speedup over CPU-based solutions when the compressed sparse row (CSR) format is used to compress data and when the advanced linear algebra subroutines, e.g., cuSPARSE [7], are used to accelerate the computations. However, a practical CT system matrix stored in the CSR format may still exceed the GPU memory size. Further, while using CSB [8], BRC [9], or CSR5 [10] can deliver tens of speedups over the CPU implementation, these formats deliver significantly lower speedup when used in ART-based CT algorithms.

In targeting ART-based CT, we propose cuART, which provides a complete compression and parallelization solution for ART-based CT on GPUs. Based on the symmetries in the system matrix, we propose a *CSR* variant dedicated for the *s*ystem matrix in CSR (*SCSR*) to further compress data by removing symmetric but redundant non-zero elements. We also leverage the sparsity pattern of X-ray projections to transform the CSR format into multiple dense sub-matrices.

Based on the optimized data structure, we then design a transposition-free computational kernel to optimize data-access patterns for SpMV and SpMV_T, avoiding the overhead of explicit sparse matrix transposition. Our major contributions include (1) the SCSR format to further compress the CSR format for the system matrix of ART and block data into multiple dense sub-matrices; (2) a transposition-free computational kernel to improve performance on the GPU by optimizing data-access patterns for both SpMV and SpMV_T; and (3) an evaluation of our approach using practical datasets on a modern compute node with the multi-core CPU and the NVIDIA Kepler GPU. Our cuART method achieves up to a 14.3-fold speedup over a single-threaded implementation on the CPU and 6.3, 6.4, and 4.3-fold speedups over multi-threaded counterparts on the GPU using cuSPARSE, BRC, and CSR5, respectively.

II. Background and related work

A. Algebra Reconstruction Technique

ART starts from an initial guess of an image vector $F^{(0)}$, then repeatedly updates the estimate image until its pixel values are convergent in some criteria. Iteration of ART can be mathematically defined as:

$$f_j^{(n+1)} = f_j^{(n)} + \lambda w_{ij}(p_i - \textstyle\sum_{j=1}^{N} f_j^{(n)} w_{ij}) / (\textstyle\sum_{j=1}^{N} w_{ij}^2) \qquad (1)$$

where $i = 1, 2, ..., M$, $j = 1, 2, ..., N$, λ is relaxation parameter, $f_j^{(n)}$ is ith pixel of estimated image after n iteration, p_i is the ith entry of projection vector, and w_{ij} is the corresponding entry of system matrix that represents weighting factor of jth pixel to the ith projection data.

Equation (1) shows each ART iteration will update the whole estimate image based on the corresponding ray and projection data; and the major computational part is the matrix-vector multiplication, i.e. $\sum_{j=1}^{N} f_j^{(n)} w_{ij}$.

B. Sparse Matrix Compression and SpMV

The system matrix is sparse because most weighting factors are zero. Sparse matrix compression and SpMV computation have been extensively studied over the past years. The compressed sparse row (CSR) format is one of the most widely-used formats that compresses matrix in row-major format and stores non-zero elements into contiguous locations. It consists of three vectors — *val*, *col_idx* and *row_ptr* — to store non-zero elements, corresponding column indices, and offsets of rows, respectively. Figure 1 (a)-(b) show an example of the CSR format. Although CSR is memory-efficient for only storing non-zero elements, it is not computational efficient, because it requires indirect addressing steps to get each non-zero elements. Other CSR variants include BRC [9], CSR5 [10], and CSB [8]. Although these variants optimize for storage and leverage hardware features of the CPU and GPU to improve the performance of SpMV, they do *not* take any specific sparsity structure into consideration.

C. GPU and GPU-based CT

In recent years, graphics processing units (GPUs) have been used to accelerate many scientific applications, e.g., [11]–[14]. Some researchers have also studied the CT image reconstruction on GPUs. For example, Zhao et al. [6] proposed a CUDA-based GPU approach with empty space skipping and multi-resolution techniques. This method calculated the weighting factors on the fly, and hence, suffered heavy computational overhead. Liu et al. [7] accelerated SART with cuSPARSE after compressing the system matrix into the CSR format. They reported that the GPU implementation was 10-fold faster than the single-threaded version on CPU at the computational kernel level. However, these reported speedup was much lower than those claimed in other sparse applications accelerated by cuSPARSE. As a result, there are potential opportunities to optimize such an application on GPU by leveraging application traits.

III. cuART Design

A. Symmetry-Based System Matrix Compression

Even though CSR is one of the most efficiently compressed formats for a sparse matrix, the data size of CT system matrix in CSR may still be much larger than the memory capacity of a commodity GPU. This motivates us to further reduce the size of system matrix by leveraging the physical characteristics of the CT scan. Specifically, we find that the X-rays may have two kinds of symmetries: *reflection symmetry* and *rotational symmetry*. Figure 2 demonstrates the symmetries among the views of X-rays: the views at position v_1 and v_3 visually have reflection symmetry, while the views at position v_2 and v_4 have rotational symmetry.

Consequently, we propose a symmetry-based CSR (SCSR). For SCSR, we first equally divides the working area into eight zones. Only rows coming from views within one zone need to be stored, and rows that come from views within any other zone can be obtained on the fly, i.e., their numerical values are the same as the stored ones following the mapping rules. Assuming an image is $n \times n$ and in a coordinate system, as shown in Figure 2; the rays within zone II are reflection symmetric to the corresponding rays in zone I, while the rays within zone III are rotationally symmetric to the rays in zone I. A pixel having the coordinate (x_1, y_1) (we denote its index as $I_1 = ny_1 + x_1$) should have corresponding pixels defined by the mapping rules:

$$\begin{cases} I_2 = ny_2 + x_2 = n(n - y_1) + x_1\text{: reflection sym} \\ I_3 = ny_3 + x_3 = n(n - x_1) + y_1\text{: rotational sym} \end{cases} \tag{2}$$

Hence we only need to store the rows coming from the views in zone I, compute indices of elements from all other rows on the fly, and then get their values. Applying this symmetry-based compression on CSR reduces memory usage to $\frac{1}{8}$ of traditional CSR.

B. Sparsity-Specific Blocking

With real image data, the longest row may have four times as many non-zero elements than others, and the distribution of non-zero elements in CSR format is irregular. In SCSR, we leverage the sparsity patterns of the system matrix to group data into multiple dense blocks.

We exploit a *global-level blocking* method to transform CSR data into dense sub-matrices. Figure 1 shows the schematic diagram of the global-level blocking. Figure 1(a) is the original system matrix, and Figure 1(b) is the corresponding CSR format. Figure 1(c) shows the rearranged non-zero elements from (b), after shifting them to the left and sorting on rows. After that, we pair rows by using the following rule: we pair the first row to the last row, the second row to the second last row, and so on. Then we concatenate each pair and pad them with zeros to get the dense matrix shown in Figure 1(d). Finally, we group non-zero elements to three dense sub-matrices, as shown in Figure 1(f): Sub_SM_1, Sub_SM_2, and Sub_SM_3. Only Sub_SM_2 and Sub_SM_3 have slight padding data; only non-zero elements in a row of Sub_SM_2 may come from different rows of the system matrix. We use an additional bitmask matrix to record the junction point, where the 1 bit indicates the change of the row ids and the 0 bit indicates the consistent row ids. For example, in the second row Sub_SM_2 of Figure 1(f), “1” comes from a different row with “7” and “8”, then the corresponding bitmask is 010. Figure 1(e) is a permutation step that make elements

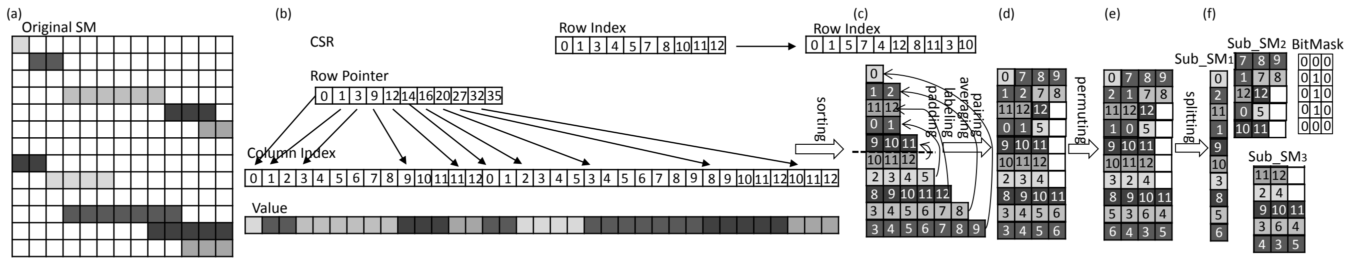

Figure 1. Schematic diagram of CSR to dense matrix based on global sparsity pattern of CT system matrix. White entries are zero paddings, numbers are column indices.

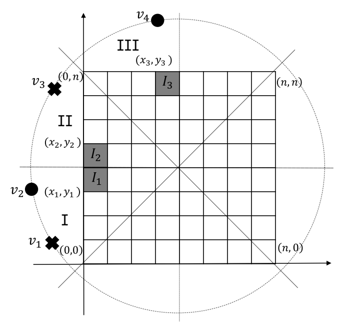

Figure 2. Schematic diagram of views/X-rays symmetries.

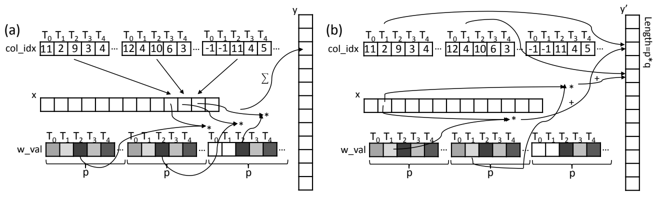

Figure 3. Data-access pattern for (a) SpMV and (b) SpMV_T.

in a same column have different values to avoid atomic operations in SpMV_T.

Compared to BRC and CSR5, our blocking scheme can reduce the overhead of atomic operations due to the grouping of non-zeros into three dense matrices. In addition, our padding overhead is less than BRC. Our SCSR does not need additional addressing steps, since compressed rows can be directly mapped from the original row indexes. Lastly only one-third of the sub-matrices, i.e., Sub_SM_2, needs the bitmask. This method has the least control divergence because each row of Sub_SM_2 has at most one junction point.

C. GPU Kernel for ART

Figure 3(a) illustrates the memory layout of the dense sub-matrix Sub_SM_3 (assume the size of this dense matrix is $p * q$) and the data-access pattern using multiple GPU threads for SpMV. Each thread is assigned to compute the product of one row and the vector x. This one-thread-to-one-row mapping can fully utilize the hardware resource. As shown in Figure 3(a), thread 2 handles row 2 in the dense sub-matrix. The values in the corresponding locations of x are calculated by using the column index. The products are accumulated and stored to the result vector y at the position equal to row 2 in this case.

When we process SpMV_T, we do not need to change the data layout. Figure 3(b) illustrates the data-access pattern for SpMV_T. Each thread still processes a compressed row. However, instead of accumulating the element-wise products within each thread, threads need to accumulate the product to the corresponding entry in vector y', whose address is the same to the column index. For example, thread 2 multiplies w_{2j} by s_2 and accumulates products to y'_2. Because we permute data inside each row to make the non-zero elements have different original column-index in one column, e.g., the first column has the value "11", "2", "9", "3", and so on, different threads accumulate the products to different positions in y' at the same time. Hence, we can eliminate or fully avoid the overhead of atomic operations.

IV. Evaluation

We evaluate cuART on an Intel Xeon E5-2697 multi-core CPU running at 2.70GHz with 64GB main memory and a NVIDIA Tesla K20x GPU. The GPU has 2688 CUDA cores and 6GB GDDR5 global memory. Our experimental data comes from *Shepp-Logan phantom* [15] and has 1024×1024 and 512×512 images. The experimental parameters of the CT device are set to 720 views and 1024 rays per view. When using the CSR format, we need 11.97GB of memory, which exceeds the global memory size of GPU. As a result, for implementations that use CSR, BRC, and CSR5 on GPU, we have to partition the test data into multiple blocks and pipeline the GPU computation and the data transfer on PCI-e. The explicit transposition from CSR to CSC for SpMV_T adversely impacts the performance because we have to pipeline the GPU computation with the data transfer on two compressed matrices.

We first compare cuART with the single-threaded implementation on CPU and the parallel implementations on GPU using CSR (cuSPARSE), BRC, and CSR5, respectively. All counterparts use the $csr2csc$ function in cuSPARSE to transpose the system matrix explicitly for SpMV_T.

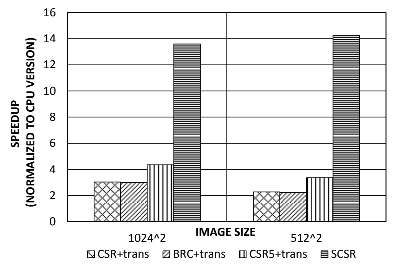

Figure 4. Performance comparisons of CSR, BRC, CSR5, and SCSR (normalized to the single-threaded implementation on CPU).

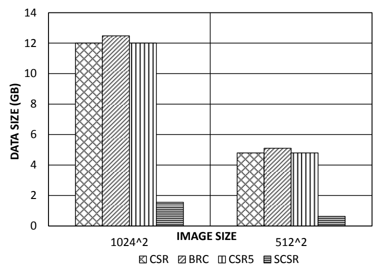

Figure 5. Memory footprint comparison in the schemes on GPU.

Figure 4 shows the normalized speedups to the single-threaded implementation on CPU. Both the CPU and GPU version run adequate iterations to reach convergence. CSR and BRC can only achieve up to a 2.9-fold speedup over the CPU version, while CSR5 can achieve up to a 4.5-fold speedup. In contrast, our SCSR can achieve up to a 14.3-fold speedup. BRC and CSR5 underperform when compared to previously reported numbers because of (1) the slower convergence speed than the CPU version, (2) the overhead from processing control tags and flags inside the data structures, (3) the time spent in the explicit transposition from CSR to CSC, and (4) the overhead in the pipeline mode to overlap the GPU computation and the data transfer on PCI-e. Except for the convergence speed, SCSR can avoid all of these overheads and improve performance significantly.

Figure 5 compares the memory footprint between the compressed data formats. Our SCSR requires *7.7 and 7.6 times less memory* than CSR for 1024^2 and 512^2 images, respectively. By leveraging the symmetry characteristics in the system matrix, SCSR can remove the symmetric but redundant non-zero elements. Also, SCSR needs much fewer zero paddings and bitmasks than BRC and CSR5. Overall, SCSR is the only format that can fit into the global memory of a single NVIDIA Tesla K20x GPU.

V. Conclusions and Future Work

In this work, we propose cuART, a fine-grained image reconstruction algorithm to accelerate ART-based computed tomography on GPUs. It consists of a SCSR data format and a transposition-free GPU kernel of ART. SCSR is a variant format of CSR that can further compress the CSR-based system matrix of CT scan and block data into several dense sub-matrices. The computational GPU kernel is designed to optimize SpMV and SpMV_T at the same time. Our experiments illustrate that cuART can reduce the memory usage and achieve much better performance compared to the counterparts both on the CPU and GPU. In the future, we seek to extend cuART to multiple nodes with an optimized MPI implementation on GPU clusters [12].

VI. Acknowledgment

This work was supported in part by the Institute for Critical Technology and Applied Science (ICTAS), an institute dedicated to transformative, interdisciplinary research for a sustainable future (http://www.ictas.vt.edu).

References

[1] R. Gordon, R. Bender, and G. T. Herman, "Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography," *Journal of theoretical Biology*, vol. 29, no. 3, pp. 471–481, 1970.

[2] H. Guan and R. Gordon, "A projection access order for speedy convergence of art (algebraic reconstruction technique): a multilevel scheme for computed tomography," *Physics in medicine and biology*, vol. 39, no. 11, p. 2005, 1994.

[3] S. Zhang, D. Zhang, H. Gong, O. Ghasemalizadeh, G. Wang, and G. Cao, "Fast and accurate computation of system matrix for area integral model-based algebraic reconstruction technique," *Optical Engineering*, vol. 53, no. 11, pp. 113 101–113 101, 2014.

[4] C. Laurent, F. Peyrin, J.-M. Chassery, and M. Amiel, "Parallel image reconstruction on mimd computers for three-dimensional cone-beam tomography," *Parallel Computing*, vol. 24, no. 9, pp. 1461–1479, 1998.

[5] F. Griill, M. Kunz, M. Hausmann, and U. Kebschull, "An implementation of 3d electron tomography on fpgas," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, Dec 2012, pp. 1–5.

[6] X. Zhao, J.-J. Hu, and T. Yang, "Gpu based iterative cone-beam ct reconstruction using empty space skipping," *Journal of X-ray science and technology*, vol. 21, no. 1, pp. 53–69, 2013.

[7] R. Liu, Y. Luo, and H. Yu, "Gpu-based acceleration for interior tomography," *Access, IEEE*, vol. 2, pp. 757–770, 2014.

[8] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. ACM, 2009, pp. 233–244.

[9] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, "An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on gpus," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 273–282.

[10] W. Liu and B. Vinter, "Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15. New York, NY, USA: ACM, 2015, pp. 339–350.

[11] X. Yu and M. Becchi, "Gpu acceleration of regular expression matching for large datasets: Exploring the implementation space," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '13. New York, NY, USA: ACM, 2013, pp. 18:1–18:10.

[12] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda, "Gpu-aware mpi on rdma-enabled clusters: Design, implementation and evaluation," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 10, pp. 2595–2605, 2014.

[13] J. Zhang, H. Wang, and W.-c. Feng, "cublastp: Fine-grained parallelization of protein sequence search on cpu+gpu," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[14] L. Li, H. Liu, H. Wang, T. Liu, and W. Li, "A parallel algorithm for game tree search using gpgpu," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 8, pp. 2114–2127, 2015.

[15] L. Shepp and B. Logan, "The fourier reconstruction of a head section," *Nuclear Science, IEEE Transactions on*, vol. 21, no. 3, pp. 21–43, June 1974.