GPU Accelerated Structure-Exploiting Matched Forward and Back Projection for Algebraic Iterative Cone Beam CT Reconstruction

Thompson, William M and Lionheart, William RB

2014

MIMS EPrint: **2016.48**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

GPU Accelerated Structure-Exploiting Matched Forward and Back Projection for Algebraic Iterative Cone Beam CT Reconstruction

William M. Thompson and William R. B. Lionheart

***Abstract*—Algebraic iterative reconstruction algorithms have been the subject of much research into problems of limited data CT reconstruction. However, their use for practical problems, particularly for high resolution systems and corresponding large volume sizes, has often been considered unfeasible due to their high computational demands. For cone beam geometry, we present a highly parallel adaptation of Siddon's algorithm for discrete forward and back projection, based on exploitation of structure in the pattern of the cone beam rays. The proposed algorithm has been implemented for nVidia GPUs using CUDA, resulting in speedup of an order of magnitude when tested against a non-structure exploiting parallel CPU implementation of Jacobs' algorithm. The work is presented in the context of circular scan, flat panel detector micro CT, but could easily be adapted to other scanning trajectories and detector configurations.**

I. Introduction

Cone beam x-ray micro CT is now a widely-used imaging technique in materials science [1], and also in other applications such as non-destructive testing, oil exploration and semiconductor manufacture for example. In such applications, there is currently a drive towards faster acquisition times, motivated by a variety of reasons including higher throughput scanning, dose reduction and higher temporal resolution four dimensional imaging. To this end, the use of algebraic iterative reconstruction algorithms involving total variation (TV) minimisation (e.g. [2], [3]) is the subject of much current research. Such algorithms have particularly high utility in these applications, as objects of interest often consist of homogeneous materials with a piecewise constant structure. A typical problem may involve calculating distribution of pore sizes based on analysis of a segmented volume, for example.

At the core of any algebraic iterative reconstruction algorithm are the projection matrix A, and its transpose, representing back projection; these are formed from a discrete model of the projection process, for which many choices exist. A common choice is to represent the rays as straight lines, and to consider the lengths of intersection with a regular grid of voxels. Then the discrete forward projection of the n^{th} ray is given by

$$p_n = \sum_{i,j,k} l_n(i,j,k)x(i,j,k), \tag{1}$$

where $l_n(i,j,k)$ represents the length of intersection, and $x(i,j,k)$ is the value of the CT attenuation coefficient function in the voxel with x, y and z indices (i,j,k). All such $l_n(i,j,k)$ together form the projection matrix A; however, due to the size of the system, these values are usually not stored but are computed on the fly during forward or back projection.

Siddon's algorithm [4] gives an efficient way to calculate the values of $l_n(i,j,k)$ by following rays through the volume. Subsequent work ([5], [6], [7], [8]) has refined Siddon's algorithm somewhat, and more recently, a new approach has been proposed [9]. However, all of these methods are presented in the general case of a single ray, and do not exploit any structure in the pattern of multiple rays.

A typical cone-beam micro CT scanner consists of a microfocus x-ray source located opposite a square flat panel detector; the object under inspection is rotated in the path of the beam to change the projection angle. Typical detector resolutions can be 2048×2048 pixels or higher, with resulting reconstructed volume size of 2048^3 voxels; therefore, in order to enable algebraic iterative reconstruction of realistically sized experimental data sets within reasonable time scales, it is essential to make the discrete forward and back projection operations as fast as possible. For this reason, in practical applications it is often the case that a different discrete model is used for back projection than that used for forward projection. In such cases, the matrix representing back projection is not the exact transpose of the projection matrix, which may have a negative impact on algorithm convergence. Our work has focused on implementations where the same model is used for both, creating a matched forward and back projector pair.

The work presented here discusses acceleration of Siddon's algorithm in the cone beam case, by exploiting structure in the rays. Although the work is presented in the specific case of a flat panel detector, the method is also equally applicable to curved detectors. The work is presented in the context of forward projection, but it should be noted that the implementation of the corresponding matched back projector uses the same method for the calculation of the ray-voxel intersection lengths, but with obvious differences in how these are used.

II. Siddon's Algorithm

Siddon noted that since most of the $l_n(i,j,k)$ are zero, rather than summing (1) over all voxels, it is much more efficient to follow the path of each ray through the volume, summing over only the non-zero values.

Considering only the two-dimensional case initially, and adopting the notation of [5], let a ray have end points

Henry Moseley X-ray Imaging Facility and School of Mathematics, University of Manchester. Corresponding author: William R. B. Lionheart, E-mail: bill.lionheart@manchester.ac.uk.

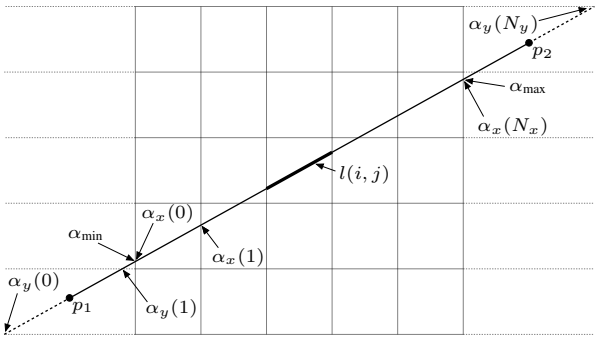


Fig. 1: Parametric representation of rays in Siddon's algorithm

$p_1 = (p_{1x}, p_{1y})$ and $p_2 = (p_{2x}, p_{2y})$, and assume $p_{1x} \neq p_{2x}$ and $p_{1y} \neq p_{2y}$; the degenerate cases can be handled trivially. Let $p = (p_x, p_y)$ be any point along the ray; then we have the parametric representation

$$p_x(\alpha) = p_{1x} + \alpha(p_{2x} - p_{1x}), \tag{2}$$
$$p_y(\alpha) = p_{1y} + \alpha(p_{2y} - p_{1y}), \tag{3}$$

where $\alpha \in \mathbb{R}$ and $\alpha \in [0, 1]$ for points on the ray between p_1 and p_2.

Without loss of generality, assume that the reconstruction grid consists of isotropic cubic voxels of unit size, with the grid origin at the lower left corner. In reality, the source and detector geometry can be simply scaled and translated to fit. Define the grid by its edges, the surfaces $x = i$ and $y = j$, where $i \in \{0, \ldots, N_x\}$ and $j \in \{0, \ldots, N_y\}$, and denote these respectively as the x and y planes.

Let $\alpha_x(i)$ and $\alpha_y(j)$ represent the α values at the intersection points of the ray with the i^{th} x plane and j^{th} y plane, as shown in figure 1; then

$$\alpha_x(i) = \frac{i - p_{1x}}{p_{2x} - p_{1x}}, \qquad \alpha_y(j) = \frac{j - p_{1y}}{p_{2y} - p_{1y}}. \tag{4}$$

Note that these points are not restricted to intersections lying within the grid.

Denote the values of α at the ray's grid entry and exit points by α_{min} and α_{max}. These may be calculated easily from the α_x and α_y values at the grid extremities. Denote the indices of the first intersected x and y planes *after the ray enters the grid* by respectively i_{min} and j_{min}. Similarly, denote the indices of the last intersected x and y planes, up to and including the grid edge, by respectively i_{max} and j_{max}. Again, these may be calculated from the α_x and α_y values at the grid extremities.

Using these calculated indices, we now form arrays $\alpha_x[.]$ and $\alpha_y[.]$ containing the α values of every intersection point of the ray with the x and y planes:

$$\alpha_x[i_{\text{min}}, \ldots, i_{\text{max}}] = [\alpha_x(i_{\text{min}}), \ldots, \alpha_x(i_{\text{max}})], \tag{5}$$
$$\alpha_y[j_{\text{min}}, \ldots, j_{\text{max}}] = [\alpha_y(j_{\text{min}}), \ldots, \alpha_y(j_{\text{max}})]. \tag{6}$$

We then sort the elements of $\{\alpha_{\text{min}}, \alpha_x[.], \alpha_y[.]\}$ in ascending order, forming the single array $\alpha_{xy}[.]$, containing the α values of every intersection point of the ray with the grid edges, in monotonically increasing order. Note that in the construction of the $\alpha_{xy}[.]$ array, duplicate values of α_x and α_y are not included. These values correspond to intersections of the ray with the corner (or in 3D, also an edge) of a voxel.

Now we loop through all points in the $\alpha_{xy}[.]$ array and calculate the x and y voxel indices and intersection lengths. For each $m \in \{1, \ldots, N_v\}$, where N_v is the total number of intersections, the x and y indices i_m and j_m are given by

$$i_m = \left\lfloor p_x\left(\frac{\alpha_{xy}[m] + \alpha_{xy}[m-1]}{2}\right)\right\rfloor, \tag{7}$$
$$j_m = \left\lfloor p_y\left(\frac{\alpha_{xy}[m] + \alpha_{xy}[m-1]}{2}\right)\right\rfloor. \tag{8}$$

Then the intersection length for each voxel is given by

$$l(i_m, j_m) = \Big(\alpha_{xy}[m] - \alpha_{xy}[m-1]\Big).d_{\text{conv}}, \tag{9}$$

where d_{conv} is the Euclidean length of the ray.

The algorithm can be generalised to 3D by following essentially the same procedure to create an $\alpha_z[.]$ array, and then forming the total sorted array $\alpha_{xyz}[.]$.

In CPU implementations of Siddon's algorithm, or any of its derivatives, we can parallelise by simply distributing the rays among available threads. However, this approach treats all rays independently, and as such does not exploit any structure. It is also not ideal for GPU implementations, since these algorithms necessitate a high level of conditional branching. This causes the GPU threads to take different execution paths, a phenomenon known as warp divergence, which can have serious consequences for performance. Memory access patterns are also unlikely to be well-structured, with similar impact on performance.

III. Structure-Exploiting Cone Beam Algorithm

Defining the cone beam geometry as in figure 2, the key to our approach for accelerating Siddon's algorithm in the cone beam case is the observation that for any projection angle and u coordinate on the detector, the projection of this ray in the z direction onto the x-y plane is completely independent of the detector v coordinate. Hence, for any such ray, the values of α_x and α_y, and the x and y voxel indices, are also independent of the detector v coordinate. Therefore, we seek to decompose the ray tracing algorithm into a component in x-y and a component in z.

As a first step, we order the volume so that z is the fastest increasing dimension, and order the projections with detector v coordinate fastest increasing. Then the loop over detector v coordinate can be brought into the core of the ray tracing algorithm, resulting in far more localised memory access patterns. This approach leads to significant performance increase, but the calculation of the ray tracing parameters in the z direction is still dependent on components in x and y, and the algorithm still needs a high level of conditional branching.

In order to make the z component completely independent, we first make the additional assumption that the cone angle is less than or equal to $45°$. By cone angle here, we mean the angle that the rays corresponding to the minimum and maximum detector v coordinates make with the x-y plane, assuming the detector is symmetric. Note that, at least in the

case of most laboratory based micro CT scanners, this is a realistic assumption. The effect of this is that, if we consider the reconstruction volume as being composed of columns of voxels in the z direction for constant x and y, then any ray will intersect either one or two voxels in each column it passes through.

Consider any ray with $p_{1z} < p_{2z}$, and let the projection of this ray in the z direction onto the x-y plane have associated $\alpha_{xy}[.]$ array as in the 2D version of Siddon's algorithm. Now consider the m^{th} entry in this array; then the parametric value of the next intersection with a z plane, α_z, and the z index of the first intersected voxel in the currently intersected column, k, are given by

$$\alpha_z = \frac{k + 1 - p_{1z}}{p_{2z} - p_{1z}}, \qquad k = \big\lfloor p_z(\alpha_{xy}[m]) \big\rfloor, \tag{10}$$

where

$$p_z(\alpha) = p_{1z} + \alpha(p_{2z} - p_{1z}). \tag{11}$$

Figure 3 shows the locations of the points $\alpha_{xy}[m]$, $\alpha_{xy}[m+1]$ and α_z along the ray in each of the two possible cases. Now let $\alpha_{\text{min}} = \min\left(\alpha_{xy}[m+1], \alpha_z\right)$; then in both cases, the intersection lengths with the intersected voxel or voxels are given by

$$l(i_m, j_m, k) = \big(\alpha_{\text{min}} - \alpha_{xy}[m]\big).d_{\text{conv}}, \tag{12}$$

$$l(i_m, j_m, k+1) = \big(\alpha_{xy}[m+1] - \alpha_{\text{min}}\big).d_{\text{conv}}, \tag{13}$$

where i_m and j_m are the x and y indices of the currently intersected column of voxels. Note that in the single voxel case, the length assigned to the second voxel is simply zero.

For rays with $p_{1z} > p_{2z}$, we have the similar formulae

$$\alpha_z = \frac{k - p_{1z}}{p_{2z} - p_{1z}}, \qquad k = \big\lfloor p_z(\alpha_{xy}[m]) \big\rfloor, \tag{14}$$

$$l(i_m, j_m, k) = \big(\alpha_{\text{min}} - \alpha_{xy}[m]\big).d_{\text{conv}}, \tag{15}$$

$$l(i_m, j_m, k-1) = \big(\alpha_{xy}[m+1] - \alpha_{\text{min}}\big).d_{\text{conv}}. \tag{16}$$

Rays with $p_{1z} = p_{2z}$ are a trivial two-dimensional sub-case and can be handled separately in situations when they arise. However, for the CT systems this work has been developed

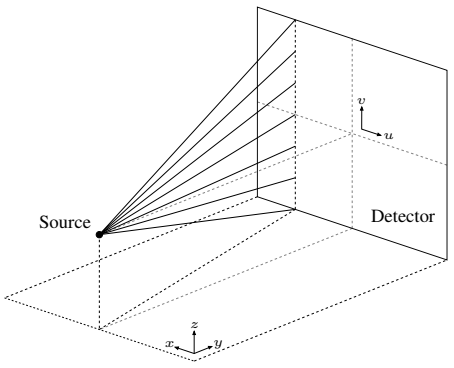


Fig. 2: The flat panel cone beam geometry, showing the exploited structure in the rays

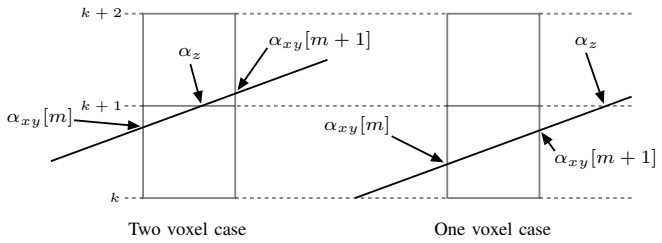


Fig. 3: Ray tracing parameters in the z direction for the one voxel and two voxel cases

for, the detector usually has an even number of pixels and this case need not be considered.

These equations form the basis of CUDA kernels for discrete forward and back projection. For each projection angle, the kernel is launched with thread blocks allocated for each detector u coordinate; each thread within a block is then assigned a detector v coordinate. The 2D $\alpha_{xy}[.]$ arrays and associated x and y voxel indices for each u coordinate are calculated as a CPU process, and copied to the GPU memory asynchronously. In practice, these arrays can often simply be pre-computed and stored for all projection angle and u coordinate pairs.

Threads in each block loop through the pre-calculated $\alpha_{xy}[.]$ array, effectively tracing the rays corresponding to all detector v coordinates simultaneously in lock-step. This process is essentially the same for forward and back projection, but the exact implementation differs in each case. At each point in the loop, the threads only access volume memory locations corresponding to the same column of voxels, keeping the memory access patterns localised. For back projection, values for the whole column of voxels are accumulated in shared memory before writing to global device memory. Multiplication by d_{conv} is performed only once; at the end for forward projection, and at the beginning for back projection. In practice, division by $p_{2z} - p_{1z}$ is replaced by multiplication by its reciprocal; since these values depend only on detector v coordinate, the values may easily be pre-computed and stored. This gives a significant performance increase, since division in CUDA maps to 16 instructions, versus a single instruction for a multiplication.

Note that since both the single voxel and two voxel intersection cases are covered by the same update equations, this results in a CUDA kernel with very few conditional branches; we need only make a decision before the main loop whether p_{2z} is greater than or less than p_{1z} to decide which set of equations to use. The vertical dimension of the detector is padded to a multiple of the CUDA warp size; hence all threads follow the same branch and warp divergence is eliminated.

Due to the limited amount of memory available on the GPU, the controlling host CPU process performs an outer loop over projection angles. For forward projection, data for the whole volume are copied to the device before starting the loop, while for back projection, it is necessary to accumulate the volume and copy this back at the end. All other host-device and device-host memory transfers are asynchronous, keeping GPU idle

	FP time (s)	BP time (s)
Jacobs, CPU, 32 threads	100	200
Inner v loop Jacobs, CPU, single thread	292	367
Inner v loop Jacobs, CPU, 32 threads	18	65
New, GPU	5.9	7.1

TABLE I: Comparison of timings for the GPU implementation

time low.

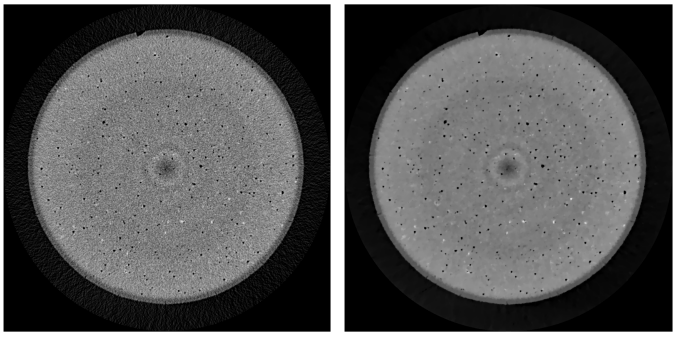
IV. Results

Our algorithm has been implemented in CUDA, and has been profiled and tuned for nVidia devices of compute capability 2.0. Table I shows timings for forward and back projection for a synthetic data test problem consisting of 720 cone beam projections of size 512×512, into a volume of size 512^3. We compared the CUDA implementation of the new algorithm against a parallel CPU implementation of Jacobs' algorithm, and serial and parallel CPU implementations of Jacobs' algorithm with inner loop over the detector v coordinate. Timings for the GPU code include the time taken for host-device and device-host memory transfers. The test hardware was a dual CPU Intel Xeon 3.1GHz machine with 64GB RAM and nVidia Quadro 6000 GPU, featuring 448 CUDA cores and 6GB onboard RAM. The operating system was Windows 7 Professional, using CUDA version 5.5. Hyperthreading was enabled.

Comparing the GPU version of our new algorithm to the parallel implementation of the Jacobs algorithm, our results show speedups of approximately 19 and 28 times respectively for forward and back projection. Comparing the new algorithm against the optimised CPU implementation of Jacobs' algorithm with inner loop over v, respective speedups of 3 and 9 times are observed. We attribute the comparatively greater speedup for back projection to the elimination of the OpenMP atomic operations which are necessary for volume updates in the parallel CPU implementations. However, back projection is still significantly slower than forward projection, since although the volume is updated in columns, the method is still essentially ray driven.

Raw performance of the algorithm in terms of the GUPS metric was measured at 40.1 GUPS for forward projection, and 28.9 GUPS for back projection. This was calculated by counting the exact number of updates made in each case to the volume or projection data, and dividing by the kernel run time as measured by the nVidia nSight profiler. While this performance for back projection falls substantially short of that given by the latest implementations submitted for the RabbitCT benchmark, it should be noted that back projection as part of a matched forward and back projector pair is a fundamentally different problem to that of a highly optimised, standalone voxel driven back projector.

The new algorithm has also been tested in algebraic iterative reconstruction methods for real experimental data sets, with varying numbers of projections of size 1024×1024 into a 1024^3 volume. Figure 4 compares a slice from an FDK reconstruction of a 400 projection scan of an Aluminium rod, at a resolution of 0.7 microns, with a reconstruction using a combined steepest descent and TV minimisation algorithm.



(a) FDK (b) Iterative

Fig. 4: Example slice from reconstruction of 400 projection experimental data set

The iterative reconstruction shows clearly lower noise level while maintaining the main features of interest, resulting in a volume which is much easier to segment. Using the new algorithm, iterative reconstruction times for this type of data set are in the order of minutes, rather than hours for our original parallel Jacobs implementation.

V. Conclusions

Due to the computation time, our previous work in application of algebraic iterative reconstruction techniques to micro CT problems has been limited to studying the effects on a central two-dimensional slice through the volume. The new algorithm presented here now allows us to apply such methods to realistic problem sizes with experimental data in full 3D, within reasonable time scales.

Our plans for future development of this work include implementations designed to cope with larger size reconstruction volumes, where the entire volume does not fit into the GPU memory, and testing and optimising the code for the latest generation of GPU devices.

References

[1] E. Maire and P. J. Withers, "Quantitative x-ray tomography," *International Materials Reviews*, vol. 59, no. 1, pp. 1–43, 2014.

[2] E. Y. Sidky and X. Pan, "Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization," *Physics in Medicine and Biology*, vol. 53, no. 17, p. 4777, 2008.

[3] X. Pan, E. Y. Sidky, and M. Vannier, "Why do commercial CT scanners still employ traditional, filtered back-projection for image reconstruction?" *Inverse Problems*, vol. 25, no. 12, p. 123009, 2009.

[4] R. L. Siddon, "Fast calculation of the exact radiological path for a 3-dimensional CT array," *Medical Physics*, vol. 12, no. 2, pp. 252–255, 1985.

[5] F. Jacobs, E. Sundermann, B. D. Sutter, and I. Lemahieu, "A fast algorithm to calculate the exact radiological path through a pixel or voxel space," *J. Comput. Inform. Technol.*, vol. 6, pp. 89–94, 1998.

[6] M. Christiaens, B. D. Sutter, K. D. Bosschere, J. V. Campenhout, and I. Lemahieu, "A fast, cache-aware algorithm for the calculation of radiological paths exploiting subword parallelism," in *Journal of Systems Architecture, Special Issue on Parallel Image Processing*, 1998.

[7] G. Han, Z. Liang, and J. You, "A fast ray-tracing technique for TCT and ECT studies," in *Nuclear Science Symposium, 1999. Conference Record. 1999 IEEE*, vol. 3, 1999, pp. 1515–1518 vol.3.

[8] H. Zhao and A. J. Reader, "Fast ray-tracing technique to calculate line integral paths in voxel arrays," in *Nuclear Science Symposium Conference Record, 2003 IEEE*, vol. 4, Oct 2003, pp. 2808–2812 Vol.4.

[9] H. Gao, "Fast parallel algorithms for the x-ray transform and its adjoint," *Medical Physics*, vol. 39, no. 11, pp. 7110–7120, 2012.