MANCHESTER 1824 The University of Manchester

GPU Accelerated Algebraic Iterative Reconstruction for Cone-Beam Micro CT

William M. Thompson and William R. B. Lionheart

University of Manchester

Introduction

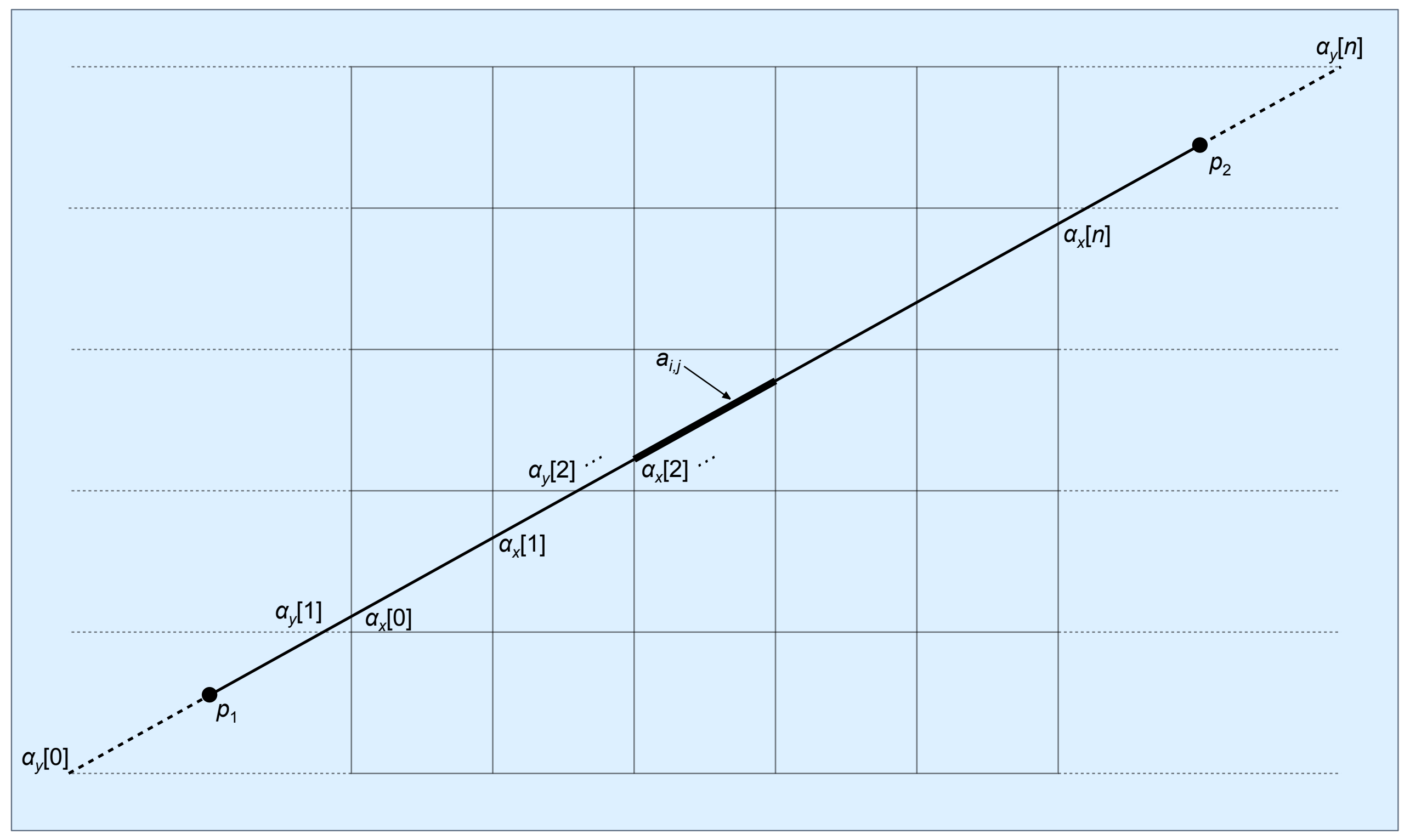
Cone-beam x-ray micro computed tomography (CT) is a high resolution three-dimensional imaging technique used in a wide variety of applications, including non-destructive testing and materials science. Generally, a divergent beam x-ray source is located opposite a flat panel x-ray detector, and the object under inspection is rotated on a stage in the path of the beam, creating a set of x-ray projections through the object at a range of angles. Analytical reconstruction algorithms for this problem are well-known [1], and perform well under ideal sampling conditions and high signal-to-noise ratios. However, in many emerging CT applications, it is desirable to reduce the x-ray dose, or reduce the data acquisition time in order to reduce motion artefacts in the reconstruction. In these cases, the data are no longer ideally sampled, and better results are often achieved with algebraic iterative reconstruction methods, which are based on the iterative solution of the system of linear equations generated from discretising the x-ray projection process.

Discretisation of the projection process is often achieved by modelling the x-ray beams as straight lines, and considering the length of intersection of these lines with voxels in the reconstruction volume. For realistic problem sizes it is impossible to simply compute and store the intersection lengths; therefore these are calculated on the fly and directly applied to the volume or projection data, in a process known as discrete forward or back projection. In order to be able to perform reconstruction in reasonable time scales, it is vital to make these processes as fast as possible.

Siddon's Algorithm

Siddon's algorithm [2] provides an efficient way to calculate the intersection lengths by following the path of each ray through the volume, using a parametric representation and considering the intersection points with the grid edges.

- Start and end points of ray are p_1 = (p_{1x}, p_{1y}, p_{1z}) and p_2 = (p_{2x}, p_{2y}, p_{2z})
- Parameter α represents normalised distance along the ray
- Arrays α_x[.], α_y[.], α_z[.] store parametric values at the intersection points with the respective x, y, z grid edges



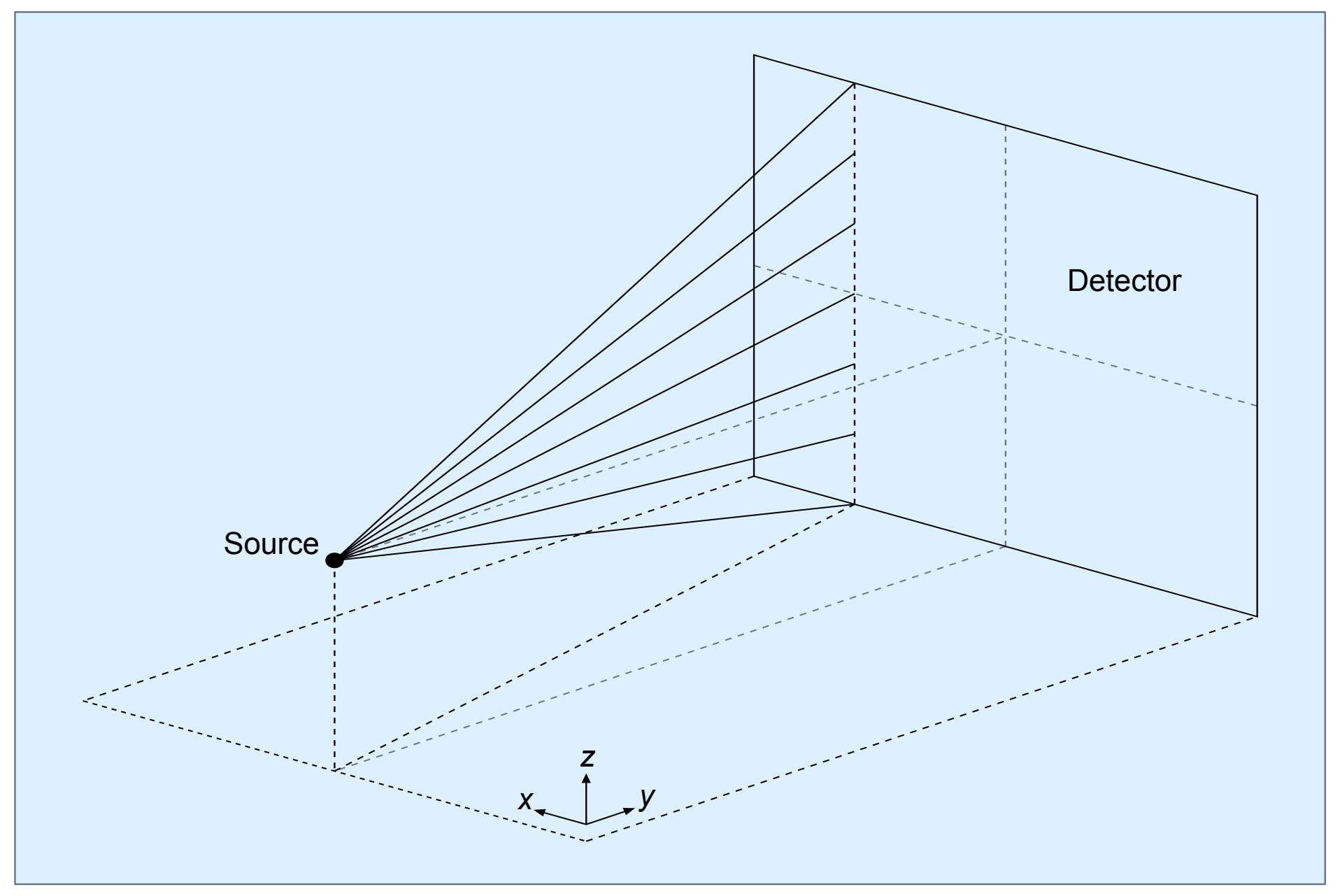
- In 2D, form array α_{xy}[.] containing sorted α_x[.] and α_y[.] values between the ray's entry and exit points into the grid
- Intersection lengths a_{ij} are then given by differences in α_{xy}[.] values, multiplied by the total length of the ray
- Voxel indices can be calculated from the α values or updated incrementally [3]

The algorithm can be generalised to 3D by including the α_z[.] values and forming the total sorted array α_{xyz}[.]. We can then parallelise this by simply distributing the rays between available threads. This approach works reasonably well in CPU implementations, but is not ideal for GPU implementation due to high levels of conditional branching within each thread, and irregular memory access patterns.

GPU Based Structure Exploiting Algorithm

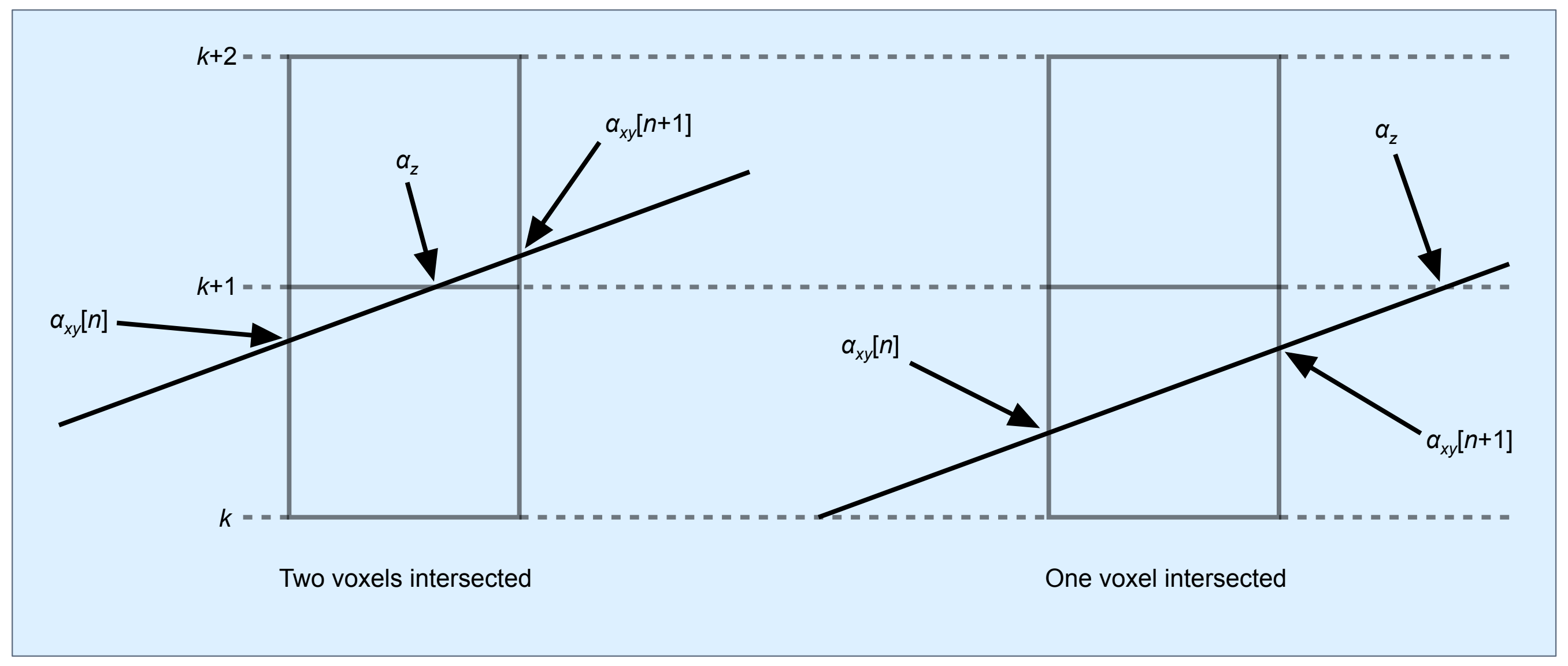
The key to our efficient algorithm is to exploit structure in the pattern of the cone-beam rays, and decompose the problem into components in the *x-y* plane and in *z*.

- For any projection angle and horizontal coordinate on the detector, rays for all vertical detector coordinates have the same projection onto the *x-y* plane
- Therefore, values of the α_{xy}[.] are independent of the vertical detector coordinate



Make the additional (realistic) assumption that the opening angle of the cone beam is less than ±45 degrees. The effect of this assumption is that any ray will intersect either one or two voxels in the *z* direction for each voxel it intersects in the x and y directions. Consider the n^{th} point along the α_{xy}[.] array for any ray with $p_{1z} < p_{2z}$, then:

- Let α_m represent the minimum of $\alpha_{xy}[n+1]$ and α_z
- Intersection length with the first voxel is given by $\alpha_m - \alpha_{xy}[n]$
- Intersection length with the second voxel is given by $\alpha_{xy}[n+1] - \alpha_m$
- In the single voxel case, zero is assigned to the second voxel
- The method is similar for $p_{1z} > p_{2z}$



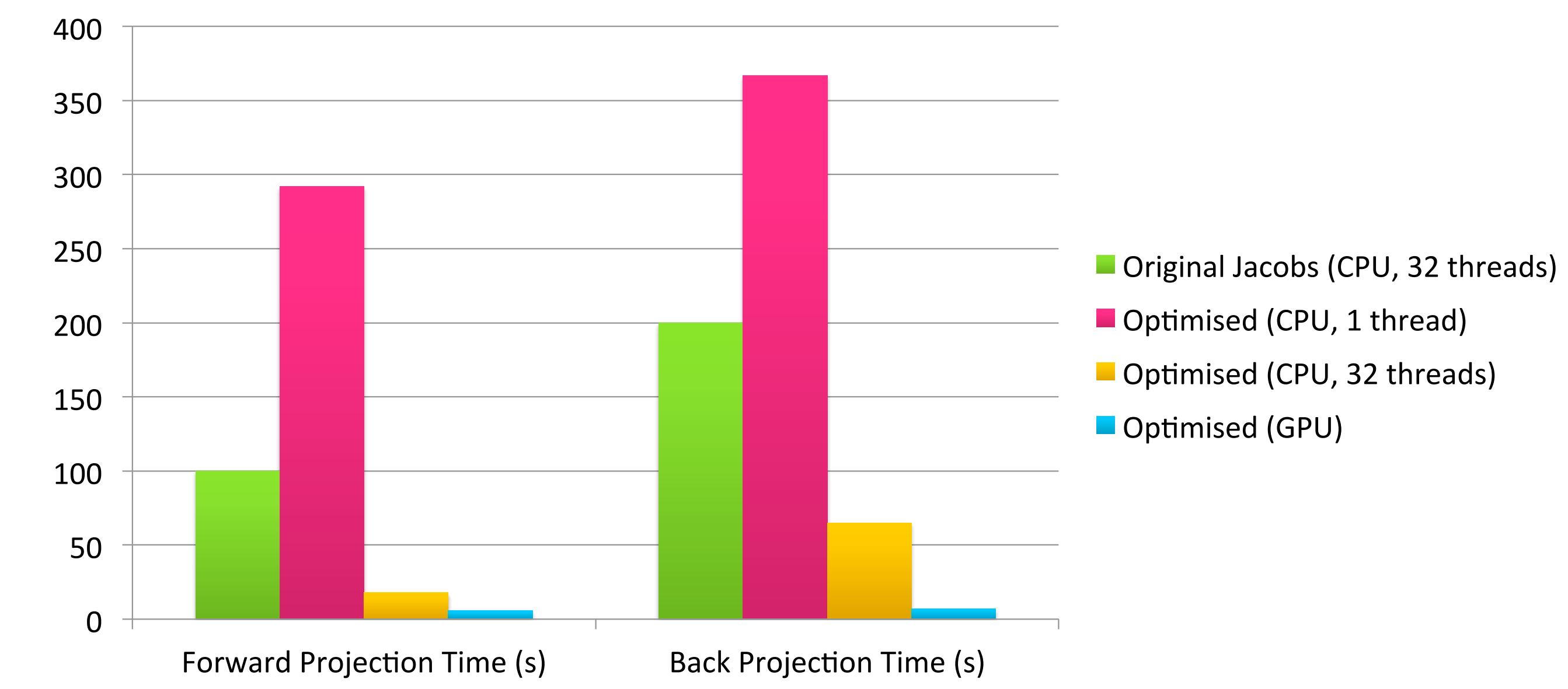
The algorithm can now be summarised as follows:

- For each projection angle, calculate α_{xy}[.] arrays and voxel indices for all horizontal detector coordinates as a CPU process
- Copy resulting arrays to GPU – due to limited memory not all can be stored
- Launch CUDA kernel on the GPU
    - Each thread in a block operates on one vertical detector coordinate
    - Kernel loops through α_{xy}[.] array, updating voxels/rays in columns
    - No conditional branching is needed
    - Memory access to projection data is fully coalesced, and to volume data is reasonably well-structured
    - For back projection, volume data is staged in shared memory
- Copy result back to host and move on to next projection angle

In practice, all memory transfer operations are performed asynchronously.

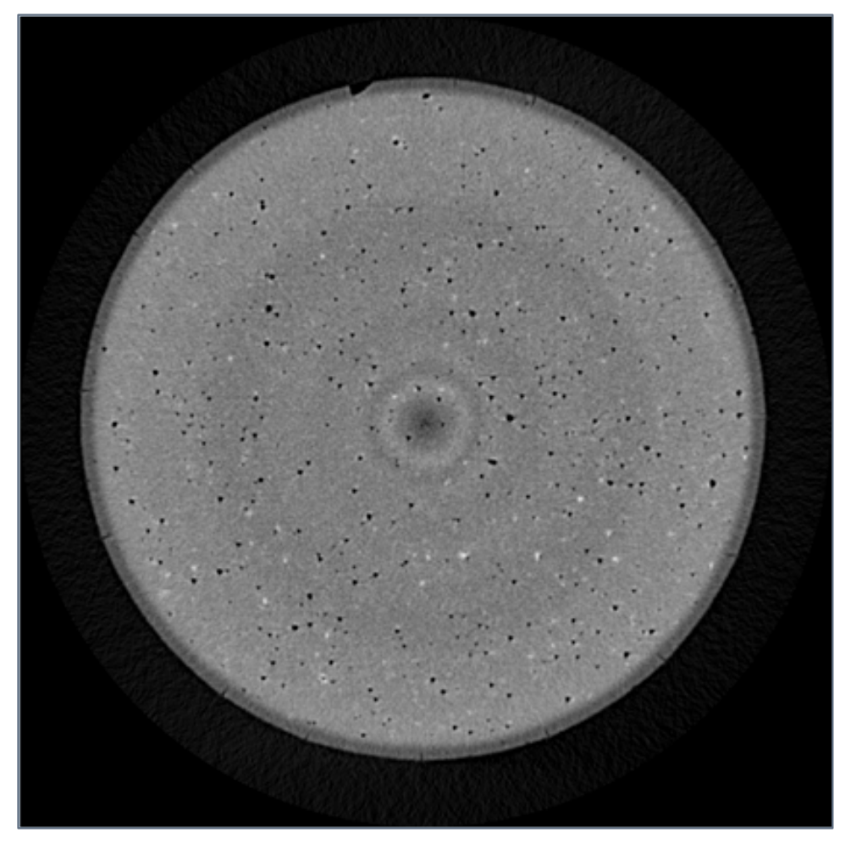
Performance Comparisons and Results

The new algorithm has been profiled and optimised for devices of compute capability 2.0. Timings are given for a test problem consisting of 720 projections of size 512^2 into a volume of size 512^3 voxels. We compare against an existing parallel CPU implementation of Jacobs' algorithm and an optimised parallel CPU version of the proposed algorithm. The test hardware was a 2x 3.1GHz Intel Xeon machine with 64GB RAM and nVidia Quadro 6000 GPU, running Windows 7 and CUDA 5.5.

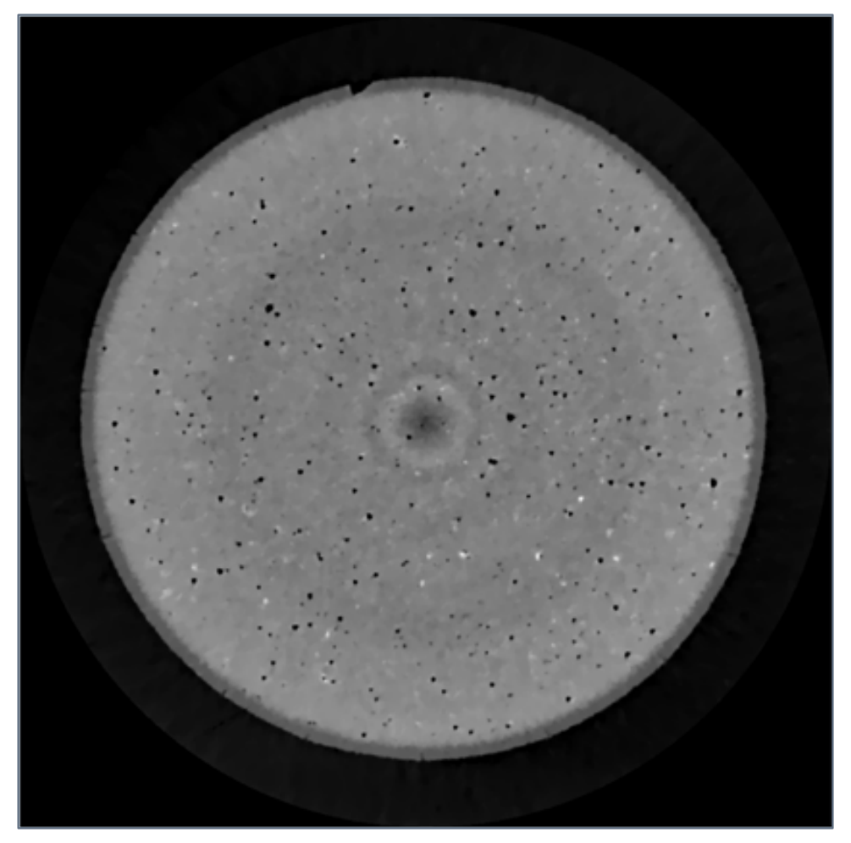


Using the new optimised GPU code, a speedup of approximately 3x is observed for forward projection, compared to the optimised CPU code running on 32 threads. A speedup of approximately 9x is observed for back projection. Note that our timings include host-to-device and device-to-host memory transfer overhead.

As an example of the intended application of the method, the images below show the results of reconstruction by FDK with a full projection set, and an algebraic iterative method using one quarter of the available data. The main features of interest in the image are maintained.



FDK



Iterative

Conclusions and Future Work

Our previous work has studied the application of algebraic iterative reconstruction techniques to problems in micro CT, but was limited to single two-dimensional slices by computational demands. Through the use of GPU technology and the new algorithm presented here, we are now able to apply these techniques to realistic problem sizes with experimental data in full 3D. This allows us to provide high quality reconstructed volumes from data sets obtained using fewer projections, enabling faster data acquisition and improved capability for imaging of dynamically varying samples. Plans for future work include testing and optimising the algorithm on latest generation nVidia GPU devices, and developing strategies for reconstructing larger dimension volumes.

References

[1] L. A. Feldkamp, L. C. Davis and J. W. Kress. Practical cone-beam algorithm. *J. Opt. Soc. Am. A*, 1(6):612–619, Jun 1984.

[2] 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.

[3] 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.*, 6:89–94, 1998.