Hardware acceleration vs. algorithmic acceleration: Can GPU-based processing beat complexity optimization for CT?

Neophytos Neophytou, Fang Xu, Klaus Mueller
Center for Visual Computing, Computer Science Department, Stony Brook University
Stony Brook, NY, USA 11794-4400

ABSTRACT

Three-dimensional computed tomography (CT) is a compute-intensive process, due to the large amounts of source and destination data, and this limits the speed at which a reconstruction can be obtained. There are two main approaches to cope with this problem: (i) lowering the overall computational complexity via algorithmic means, and/or (ii) running CT on specialized high-performance hardware. Since the latter requires considerable capital investment into rather inflexible hardware, the former option is all one has typically available in a traditional CPU-based computing environment. However, the emergence of programmable commodity graphics hardware (GPUs) has changed this situation in a decisive way. In this paper, we show that GPUs represent a commodity high-performance parallel architecture that resonates very well with the computational structure and operations inherent to CT. Using formal arguments as well as experiments we demonstrate that GPU-based 'brute-force' CT (i.e., CT at regular complexity) can be significantly faster than CPU-based as well as GPU-based CT with optimal complexity, at least for practical data sizes. Therefore, the answer to the title question: "Can GPU-based processing beat complexity optimization for CT?" is "Absolutely!"

Keywords: computed tomography, CT, 3D reconstruction, filtered backprojection, inverse Radon Transform, programmable graphics hardware, GPU

INTRODUCTION

This paper builds on our previous work [6, 9] which devised various techniques for the GPU-acceleration of a rich set of CT algorithms, including analytical (Feldkamp FDK[3]) as well as iterative (SART, OS-EM) methods, the latter with attenuation correction and detector blurring for emission tomography. In the current work, our focus is on contrasting the speedups that can be gained by GPU-acceleration of the standard backprojection operations with those that can be gained by algorithmic acceleration aimed at reducing and optimizing the theoretical computational complexity. This is a new contribution, which will help theoreticians and practitioners alike to assess the power of commodity hardware acceleration for practical applications. Our research indicates that programmable commodity graphics hardware (GPU) offers such tremendous arithmetic acceleration capabilities for standard CT that it is able to outpace any complexity-reducing algorithm, both for CPU and for GPU implementations of these. In this study, in order to maintain a fair competition, we only compare single-CPU with single-GPU solutions, noting that both platforms scale equally well in terms of the number of processors (at least for the task of backprojection).

The computational complexity for reconstructing an N^3 dataset using standard back-projection of $\pi \cdot N/2$ single-orbit projections of size N^2 is O(N^4). This complexity can be reduced to O($N^3 log N$) if one performs a polar to Cartesian regridding in the Fourier space, followed by an inverse FFT. Algorithmically-speaking, this FFT-based approach is a divide and conquer (d&c) acceleration. Computationally more involved is Grangeat's groundbreaking mathematical framework[4] for exact computed tomography (CT) from cone-beam data, using the first derivative of the Radon Transform. Although inherently of complexity O(N^5), by using the decomposition due to Marr's method final complexity could be lowered to O(N^4). Since then, various researchers[1,2] have proposed to improve this complexity even further, also to O(N^3 $log N$), by exploiting the FFT or other d&c schemes for backprojection (as well as for filtering). However, in particular the non-FFT d&c methods incur substantial overhead for data management, which can lower the overall impact that the complexity reduction has on computational speed.

All of these complexity-reduction approaches have mainly been implemented on CPUs (although DSPs could be readily used for the FFTs). But let us assume that all one has is a high-end commodity PC (as is standard in any research lab), equipped with a matching GPU board, such as an NVidia GeForce 8800 GTX (which is available for less than $500 in any computer outlet). The GeForce 8800 GTX offers 128 parallel stream processors at IEEE floating point precision (but note that the number of pipelines keeps increasing at 6-month intervals). Thus, the maximal degree of parallelism, given proper programming, is 128 (for the GeForce 8800 GTX). For now, let us further assume, for simplicity, that each pipeline is as fast as a single-core CPU, which is inherently a single-pipeline architecture (unless MMX or SSE are employed). The speedup given by arithmetic optimization is $N/\log(N)$, which is 56 for N=512 and 102 for N=1024 (larger N are currently infrequent). Thus, just using these admittedly simple arguments, GPU acceleration provides potential speedup factors better or equivalent to algorithmic optimization. But now let us take this argument further and consider the arithmetic pipelines themselves (before we just assumed that both pipelines had equal speed). The main operations in backprojection are 3D (voxel) projections, 2D (image) interpolations, and scalar (voxel) value additions. These are in fact the core operations in computer graphics algorithms, and they are thus well optimized on graphics boards, which were developed to generate high-quality real-time 3D imagery for computer games. In contrast, for general-purpose CPUs these operations form just another (completely un-optimized) sequence of instructions. Therefore it is likely that the acceleration factor of 128, as estimated above, is even higher, and the result section of this paper offers measured data to support this point.

We have shown in our previous work that the simple streaming data model of a backprojection accelerates very well on GPUs. While a divide & conquer algorithm can also be implemented, it requires more elaborate data management, which slows down the data flow and processing. Given the complexity of the hardware, contrasting these two configurations is difficult using theoretical arguments of the type employed above. We therefore resort to base our analysis on measured results.

METHOD

We use the exact reconstruction method due to Grangeat and Marr, because it is regarded the gold standard of CT. The method is based on a set of 2D filtered backprojections, one on the meridian planes of the Radon derivative data, and a subsequent one on the axial slices of the first result. The latter set of backprojections then results in the final reconstructed volume. This method will constitute our standard 'brute force' reconstruction scheme, which has the same complexity as FDK, that is, $O(N^4)$. As a means to optimize the computational complexity, Axelsson and Danielsson[1] have devised an ingenious scheme that employs various frequency-space operations mixed with FFTs to achieve a complexity of $O(N^3 logN)$. This approach will form our optimal-complexity reconstruction scheme in our study.

GPU-accelerated schemes generally tend to capitalize on the massive processing capabilities of the GPU. Any problem that can be expressed as a set of simple parallel operations can gain tremendous speedups when ported to the GPU. 2D backprojection, one of the basic operations used in exact 3D reconstruction can be easily expressed as a set of streaming texture-mapping operations and therefore accelerate well on the GPU[6, 9]. On the other hand, d&c algorithms require more overhead for program flow and therefore achieve lower speedups when programmed on GPU hardware. Accelerating the FFT on GPUs has been widely studied [5,7,10] in recent years, and a 10-fold acceleration over FFTW [11] (which is the fastest CPU implementation of the FFT) could be achieved. However, the degree of acceleration grows with the size of the data, and in fact most CT datasets are too small to take advantage of the GPU-based FFT-acceleration. On the other hand, we [6,9] and also others [7] have shown that backprojection can be sped up by 1-2 orders of magnitude on the GPU, for the reason mentioned in the Introduction.

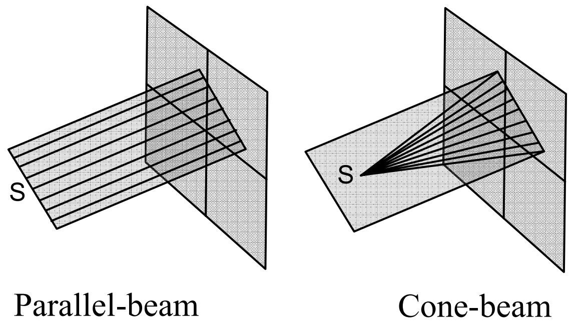

Figure 1. Comparing plane integration on parallel and cone-beam projection data.

Mapping the Grangeat Algorithm to the GPU

We base our standard 'brute-force' implementation on the description provided by Grangeat[4]. The framework consists of two stages as most "exact reconstruction algorithms" pipelines do. First the detector data is converted into Radon data, and then an inversion process is applied to the Radon data to acquire the final reconstructed volume. Because in a cone-beam acquisition setup it is not possible to directly acquire Radon data, Grangeat describes the fundamental relationship between the derivative of a line integral of cone-beam projection data and the first derivative of the Radon transform. This relation/equation is used for reconstruction from cone-beam data, since the line integral does not give the integral over a plane needed by the Radon transform.

As illustrated in Figure 1, integrating along a line in the projection data does not give the integral over the actual plane of absorption. Instead, it gives a weighted integral where the weight is proportional to *1/SA*, where *SA* is the distance from the source to a point *A* on the detector plane. This is because the rays are closer to each other near the source. Differentiating the same line integral with respect to the slice angle *θ*, gives a result weighted by *SAcosθ*, effectively canceling the effect of the original scaling in the acquisition data. The second step of the process will then reconstruct from derivatives of Radon data.

From detector data to Radon derivatives

We present some of the details of the Grangeat method in order to better describe the main parts of our GPU accelerated pipeline. For a complete description, however, we refer the reader to the original paper[4]. Let us first introduce the fundamental relationship between the derivatives of the acquisition data and the first derivative of the Radon transform of the data.

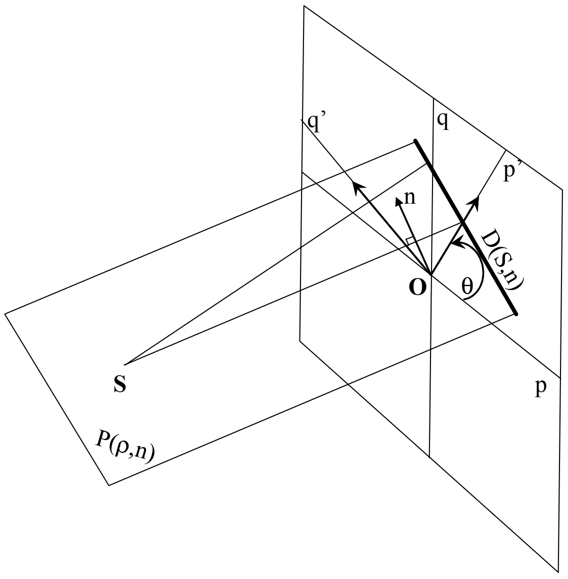

Figure 2. Cone-beam geometry as referenced in the formulas.

$$Yf(S,A) = \frac{\|SO\|}{\|SA\|} Xf(S,A) \tag{1}$$

$$R'f(\overrightarrow{OS}.\vec{n},\vec{n}) = \left(\frac{\|OS\|}{\cos\theta}\right)^2 \int_{q'=-\infty}^{+\infty} \frac{\partial}{\partial p'} Yf(S,A(q'))dq' \tag{2}$$

In the equations above,

- *R'f* is the first derivative of the Radon transform of function *f*,
- $\overrightarrow{OS}$ is the vector starting from the center of the object, going to the source of the projection,
- $\vec{n}$ is the unit vector which is the normal of the plane *P(ρ,* $\vec{n}$ *)*, with *(ρ,* $\vec{n}$ *)* being spherical coordinates. These spherical coordinates may be also represented as *P(ρ, φ, θ)*, with $\vec{n} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ and $\overrightarrow{OS}.\vec{n} = \rho$.
- *p'* and *q'* are pixel indices along the coordinate system aligned to the intersection of the current plane *P(ρ,* $\vec{n}$ *)*, and is essentially a rotation of the detector plane by the angle *θ*, formed between the intersection line *D* of the plane *P(ρ,* $\vec{n}$ *)* with the detector plane, and the horizontal direction *p* of the detector plane.
- *Xf(S,A)* is the corresponding cone-beam ray integral through the given volume that passes through the source point *S*, and terminates on the detector plane at point *A*.
- *Yf(S,A(q')* is the scaled cone-beam ray integral that terminates on the detector plane at point *A(q')*. Here, the function is differentiated with respect to the *p'* direction vector.
- *A(q')* is the corresponding point on the line *D*, for a given horizontal index *q'*.

Let us define the set of intersection lines *D* on the projector plane using polar coordinates *(r, θ),* with *r* in [*-diagonal ... diagonal*] and *θ* in [*-π/2...π/2*]. Here, *θ* is the angle between the plane horizontal direction *q* and the current *q'*. Thus, the differentiation along *q'* can be decomposed into a combination of horizontal and vertical differentiation, where the horizontal and vertical gradients are computed once and reused for the computation of all the directional derivatives.

$$\frac{\partial}{\partial q'} Yf(S,p,q) = \cos\theta \frac{\partial}{\partial p} Yf(S,p,q) + \sin\theta \frac{\partial}{\partial q} Yf(S,p,q) \qquad (3)$$

The fundamental relation outlined above, defines the first steps of the process of converting the incoming detector data into the first derivative of Radon transform that can then be used for the reconstruction of the 3D function. In our GPU pipeline this relationship is implemented via texture mapping and rotation operations applied to the input data, which are also encoded in a set of textures residing in GPU memory.

For each detector plane, with the detector angle *n* in [0...2π], the following steps are applied to convert all its points into Radon transform derivative data:

1. Store detector plane in a texture, where each point *A* is indexed in (*p*,*q*) coordinates on the plane.
2. Run a texture processing pass with a fragment program set with multiple render targets (MRT) to output into two temporary textures, G_x and G_y. The program will
    a. scale each point by $\frac{\|SO\|}{\|SA\|}$
    b. compute the derivative along *x* direction and store it in *textureOutput0* (G_x)
    c. compute the derivative along *y* direction and store it in *textureOutput1* (G_y)
3. For the set of angles *θ* in *[-π/2 ... π/2*]
    - map a polygon rotated by *θ* onto the screen.
    - apply parallel reduction to compute the sum of each line of the rotated polygon, compute the partial derivative at each point using G_x, G_y, according to equation 3. This is equivalent to applying a loop for the set of lengths *ρ* in [$-N\sqrt{2}$ ... $N\sqrt{2}$] (with *N* the size of one side of the detector plane)
    - store the result into a column vector as part of a texture that is indexed by *(θ,ρ)*.

After this process is completed for all given detector planes, the result is a new array of textures, forming a data-structure indexed by *(n,θ,ρ)*. A re-binning step is required in order to convert this data structure into the desired radon derivative, *R'(φ,θ,ρ)*, from which we can then reconstruct the final dataset. Using the description by Grangeat[4] this is equivalent to a shift of the *φ* angle for every point by $-\arcsin\left(\frac{\rho}{\|SO\|\cdot\sin\theta}\right)$ for every point of each detector plane.

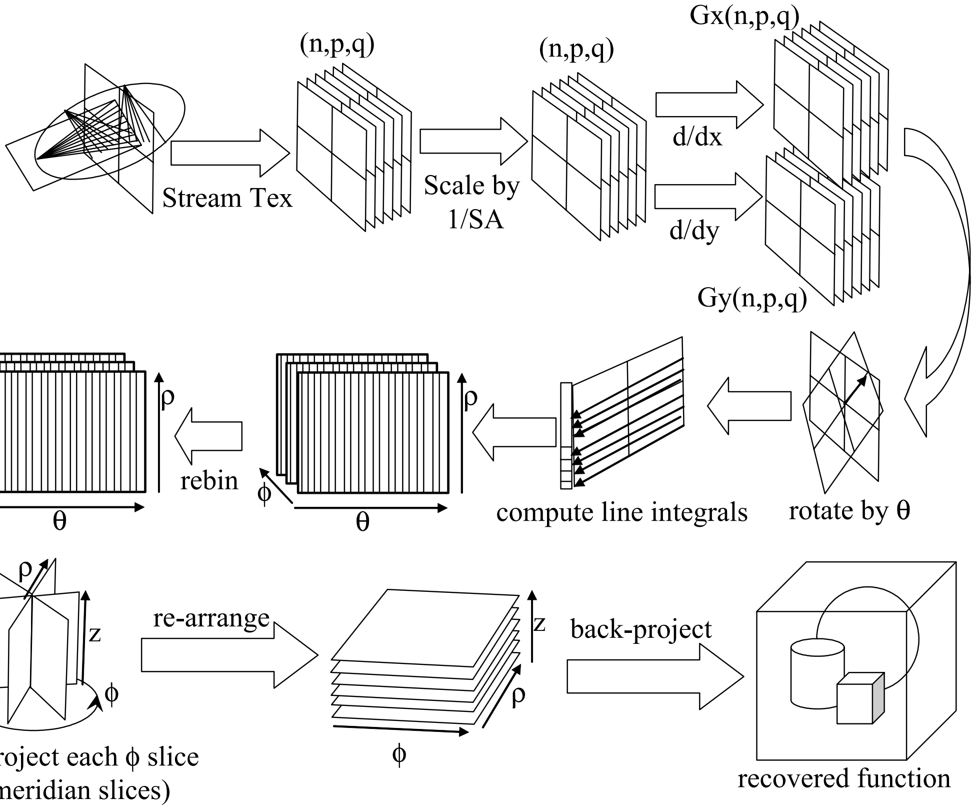

Figure 3: Overview of the reconstruction process using the Grangeat method.

From radon derivatives to the reconstructed object function

Given the first derivative of Radon transform of the function, *R'f(φ,θ,ρ)* each point *M* can be reconstructed using the following 3D inversion back-projection formula:

$$f(M) = -\frac{1}{8\pi^2}\cdot\int_{\theta=-\pi/2}^{\pi/2}\int_{\phi=0}^{2\pi}\frac{\partial}{\partial p}R'f(\phi,\rho,\theta)\cdot|\sin\theta|\cdot d\theta\cdot d\phi \qquad (4)$$

Naturally, a direct evaluation of this formula for each of the N^3 voxels of the volume would yield a wasteful O(N^5) implementation. Instead, we follow the decomposition approach outlined by Marr[12] and also described by Grangeat[4]. We also see that the decomposition approach can also be efficiently implemented on the GPU.

Using this approach, we first apply a filtered back-projection on all the meridian slices, where each reconstructed point on every slice ϕ describes the function $\tilde{X}f(\phi,B)$:

$$\tilde{X}f(\phi,B) = -\frac{1}{4\pi^2}\cdot\int_{\theta=-\pi/2}^{\pi/2}\frac{\partial}{\partial p}R'f(\phi,\rho,\theta)\cdot|\sin\theta|\cdot d\theta \qquad (5)$$

This back-projection operation is implemented as a differentiation step, followed with a series of repeated texturing operations. As the latest hardware (GeForce 8800) supports full floating point blending, the results of the repeated back-projections are left to accumulate on the same target texture, using an additive blending function. This saves about 30% of the additional overheads that would have been imposed if an accumulation scheme switching between two temporary buffers was necessary (ping-pong buffering), as in the previous generation of graphics cards.

Each of the ϕ slices of the resulting function $\tilde{X}f(\phi,B)$, now constitutes of a set of lines, where each line represents a parallel projection/integration of the object function *f* along the projection angle ϕ. Therefore, back-projecting the axial slices of function $\tilde{X}f(\phi,B)$ will result in reconstructing the slices of the object function *f*.

$$f(\mathrm{M}) = \frac{1}{2}\cdot\int_{\phi=0}^{2\pi}\tilde{\mathrm{X}}f\left(\phi, projectOntoSlice(\phi,M)\right)d\phi \qquad (6)$$

In practice, this final step can also be implemented as a simple set of back-projection operations taking advantage of the hardware's blending and texturing capabilities. An illustration of all the steps described in this section is provided in Figure 3.

The Axelsson algorithm on GPU

The algorithm proposed by Axelsson et al. [1] follows the two stage method designed by Grangeat, where the first derivative of Radon data is obtained from the cone-beam data before it is used to reconstruct the 3D object. For the first stage multiple line integrals in the detector space have to be computed to yield the weighted plane integral, which consists of all points in the 3D Radon space (see equation 2). Instead of performing this in the spatial domain, Axelsson's method converts the detector data into frequency space via a 2D FFT on each projection. After the result is re-sampled onto a polar grid, the derivative operation is applied on the object data distributed upon the polar grid. The plane integrals are obtained via a series of 1D radial FFTs. This algorithm is based on the Fourier Slice Theorem, where 1D Fourier transform of a projection (integral) of a 2D object function is equivalent to a slice of its 2D Fourier transform. The resulting data has to be re-binned using the formulas given in Section 2.1.1. Since the complexity of a 2D FFT is O($N^2 logN$), the algorithm reduces the complexity of the first step from O(N^4) to O($N^3 logN$).

In the second stage, a similar scheme is applied and another series of radial FFTs is employed for the Fourier transform of the Radon derivative data. Linogram sampling is used to compensate for the non-uniform sampling on the polar grid, where each radial line on the vertical plane is sampled at a rate determined by the rotation and elevation angles. Then, an inverse 2D FFT transform is applied to each of the vertical planes yielding a similar set to the meridian planes of the Grangeat approach described above. A second inverse 2D FFT is applied on the resulting horizontal planes, finally recovering the object function. The complexity of the second stage remains the same and the overall complexity of the method is kept at O($N^3 logN$).

In the above algorithm the dominant operators are apparently the 1D and 2D FFT transforms. We use Sumanaweera's description[8] for the FFT operator used in our implementation.

Following are some of the implementation details of the above algorithm on the GPU platform. The first stage of the algorithm proceeds as follows:

a. Pre-weight the detector data by 1/*SA*. This is a pixel-by-pixel operation accelerated on the GPU as a simple texture processing operation.

b. Apply 2D FFT transform to each weighted detector slice (projection).

c. Re-interpolate the detector data in the Fourier space from Cartesian grid to polar grid. This is done by binding the Cartesian grid into a 2D texture and using a fragment shader to render it into a sinogram texture that represents the polar grid and indexed by *(ρ,θ)*. The fragment shader consists of a coordinate transform equation and a resampling kernel to index the Cartesian texture and generate the values for each point on the polar gird.

d. Multiply each point on every slice by the constant $j2\pi R$ to obtain derivatives (in a similar way to **a**).

e. A series of inverse 1D radial FFTs are used for computing plane integrals, one for each line θ of every slice. At this stage, all the 1D FFTs of a slice can be performed in parallel using a framework similar to 2D FFT.

f. All slices are then post-weighted according to equation 2.

In the second stage of the reconstruction, the 1D Fourier transform has to be applied to each line separately due to the non-uniform sampling imposed by the linogram scheme. The subsequent 2D FFTs are also performed on GPU. Before the second 2D FFT takes place a reshuffling of the data is required, in order to transfer the data from the vertical planes and create a set of horizontal planes to be used in the second reconstruction.

RESULTS

We implemented Grangeat[4] and Marr's[12] approach for reconstruction using Radon derivatives on the GPU and compared it with our implementation of Axelsson's[1] algorithm using the Fourier Transform on both the CPU and the GPU. We based our GPU FFT implementation on Sumanaweera's[9] description, while the CPU implementation uses the FFTW library[11]. For the GPU back-projection operations we used a multi-pass texture-based approach which employs frame-buffer objects and takes advantage of the full 32-bit floating point blending capabilities of the new GPU hardware. The same compositing scheme was also used for the insertion of radial 1D data into the 2D buffer for each slice before the FFT in our Axelsson GPU implementation.

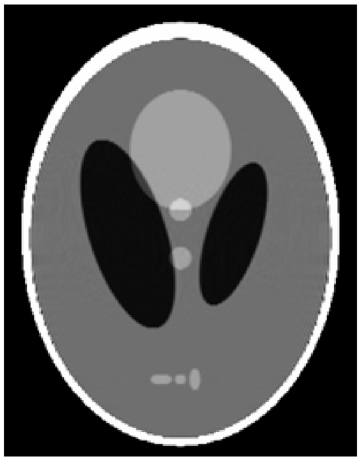
Figure 4: Reconstructed Shepp-Logan phantom

The current GPU framework directly processes projection data into Radon derivatives on the GPU, however, it has to resort to the CPU for the operation of re-binning the data into a complete data structure of the $R'(\varphi,\rho,\theta)$ function. This highly scattered operation requires capabilities that are still not implemented efficiently on the latest generation of graphics hardware. After the data is re-binned, the Radon derivatives array is loaded back to GPU memory to be used for the second stage of reconstruction using Marr's and Axelsson's approach.

We measured the total reconstruction costs for the compared techniques on a dual core AMD Athlon at 2.2Ghz, equipped with a NVidia GeForce GeForce 8800GTX GPU, excluding the costs of transferring the derivative data to and from the GPU. High-quality reconstruction results from both GPU back-projection and GPU Axelsson methods are very similar (see Figure 4 for the former). The graph of Figure 5 shows clearly that the GPU-accelerated back-projection is significantly faster than the complexity-optimized CPU-based Axelsson method for all measured sizes, with speedups ranging from 7.5 to 2.5. Finally, Figure 6 illustrates that GPU-accelerated back-projection is faster than a GPU-accelerated Axelsson method for grid sizes of less than about 1024, which is the break-even point. As expected, the Axelsson O($N^3 logN$) method eventually outperforms the O(N^4) back-projection algorithm for sufficiently large input grid sizes. However, in the context of current CT reconstruction task magnitudes, a volume size of about 1024^3 is probably sufficient in most clinical applications. The results are also consistent with the fact that current 1D FFT implementations are much better accelerated for very large input data sizes.

CONCLUSION

GPUs are perfectly suited to accelerate the most compute-intensive operation of CT, i.e., the back-projection operator. On the other hand, they are less suited to accelerate divide & conquer schemes designed for reducing the theoretical computational complexity. Our experiments also indicate that the dataset size at which the GPU-implementations of these become dominantly efficient exceed the dataset sizes encountered today in clinical practice. While this number will certainly grow over time, so are the capabilities of GPUs, and due to this fact, we feel certain that our study has a lasting impact.

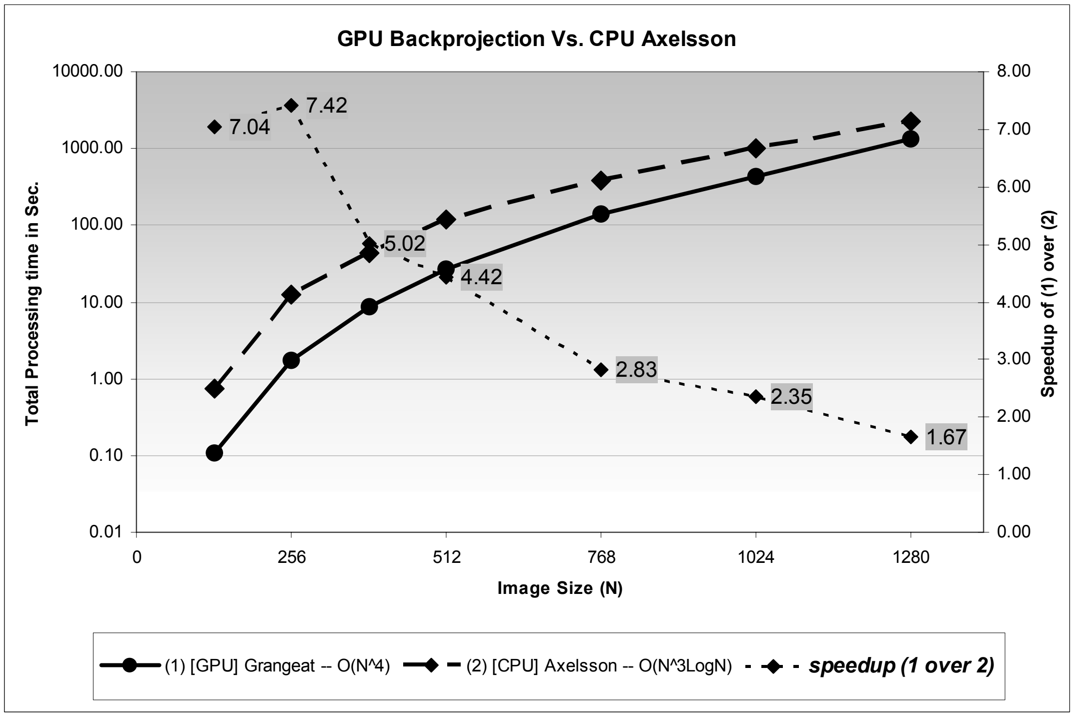

Figure 5: Total reconstruction timings for GPU Backprojection vs. CPU Axelsson

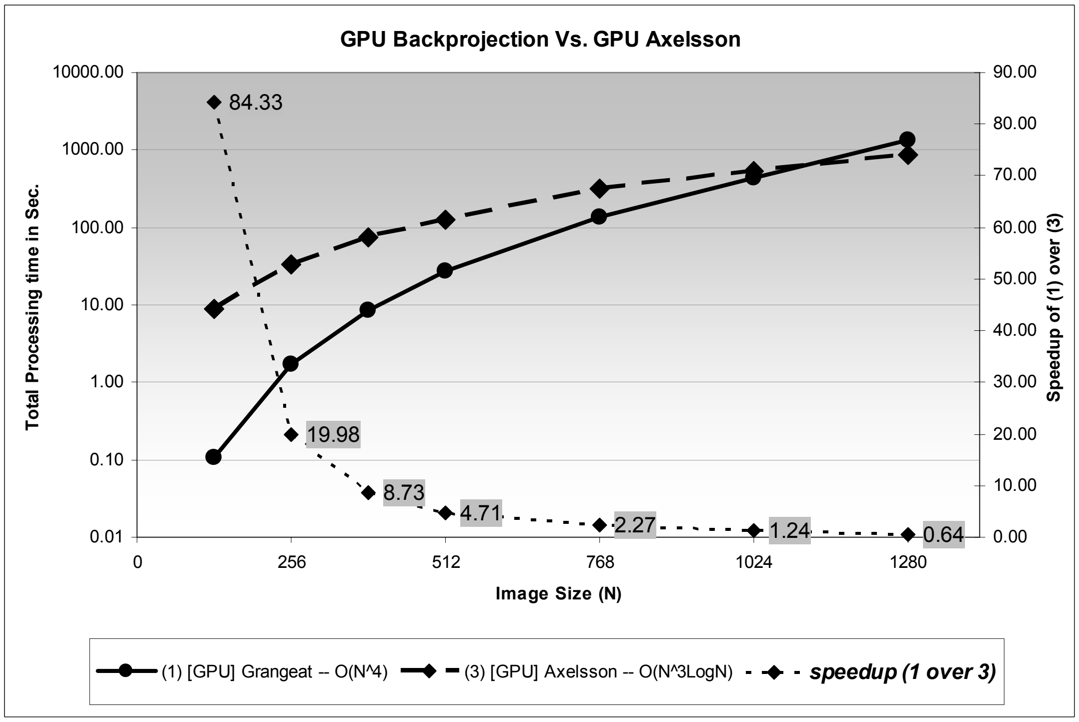

Figure 6: Total reconstruction timings for GPU Backprojection vs.. GPU Axelsson

REFERENCES

1. C. Axelsson, P. Danielsson, Three-dimensional reconstruction from cone-beam data in O(N^3logN) time, *Physics in Medicine and. Biology*, 39, 477-491, 1994.
2. S. Basu and Y. Bresler, O(N^3logN) backprojection algorithm for the 3D Radon Transform, *IEEE Trans. on Med. Imaging*, 21(2), 76-88, 2002.
3. L. Feldkamp, L Davis and J. Kress, Practical cone beam algorithms, *J. Opt Soc. Am*. A(6), 612-619, 1984.
4. P. Grangeat, Mathematical framework of cone beam reconstruction via the first derivative of the Radon transform, Mathematical *Methods in Tomography (Lecture Notes in Mathematics),* 1991.
5. T. Jansen, B. von Rymon-Lipinski, N. Hanssen, E. Keeve, Fourier volume rendering on the GPU using a split-stream-FFT, *VMV '04*, 395-403, 2004.
6. K. Mueller and F. Xu, Practical considerations for GPU-accelerated CT, *IEEE International Symposium on Biomedical Imaging (ISBI '06),* 2006.
7. T. Schiwietz, T. Chang, P. Speier, R. Westermann, MR image reconstruction using the GPU, *SPIE Med. Img., 2006.*
8. T. Sumanaweera, D. Liu, Medical image reconstruction with the FFT, *GPU Gems II*, Addison Wesley, 765-784, 2005
9. F. Xu and K. Mueller, Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware*, IEEE Trans. on Nuclear Science*, 52(3), 654-663, 2005.
10. http://gamma.cs.unc.edu/GPUFFTW/
11. http://www.fftw.org/
12. Marr, R. B., Chen, C., and Lauterbur, P. C., On two approaches to 3D reconstruction in NMR zeugmatography. *Proc. Mathematical Aspects of Computerized Tomography* (Oberwolfach) 1980.