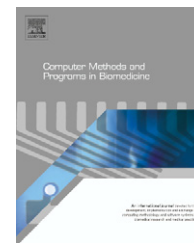




ELSEVIER

journal homepage: www.intl.elsevierhealth.com/journals/cmpb

On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs

Fang Xu^a, Wei Xu^b, Mel Jones^b, Bettina Keszthelyi^b, John Sedat^b,
David Agard^b, Klaus Mueller^{a,*}

^a Center for Visual Computing, Computer Science Department, Stony Brook University, NY 11794-4400, United States

^b Howard Hughes Medical Institute and the Keck Advanced Microscopy Center, Department of Biochemistry & Biophysics, University of California at San Francisco, CA, United States

ARTICLE INFO

Article history:

Received 25 February 2009
Received in revised form
1 September 2009
Accepted 3 September 2009

Keywords:

Iterative reconstruction
Computed tomography
Commodity graphics hardware
GPU

ABSTRACT

Expectation Maximization (EM) and the Simultaneous Iterative Reconstruction Technique (SIRT) are two iterative computed tomography reconstruction algorithms often used when the data contain a high amount of statistical noise, have been acquired from a limited angular range, or have a limited number of views. A popular mechanism to increase the rate of convergence of these types of algorithms has been to perform the correctional updates within subsets of the projection data. This has given rise to the method of Ordered Subsets EM (OS-EM) and the Simultaneous Algebraic Reconstruction Technique (SART). Commodity graphics hardware (GPUs) has shown great promise to combat the high computational demands incurred by iterative reconstruction algorithms. However, we find that the special architecture and programming model of GPUs add extra constraints on the real-time performance of ordered subsets algorithms, counteracting the speedup benefits of smaller subsets observed on CPUs. This gives rise to new relationships governing the optimal number of subsets as well as relaxation factor settings for obtaining the smallest wall-clock time for reconstruction—a factor that is likely application-dependent. In this paper we study the generalization of SIRT into Ordered Subsets SIRT and show that this allows one to optimize the computational performance of GPU-accelerated iterative algebraic reconstruction methods.

© 2009 Elsevier Ireland Ltd. All rights reserved.

1. Introduction

The rapid growth in speed and capabilities of programmable commodity graphics hardware boards (GPUs) has propelled high performance computing to the desktop, spawning applications far beyond those used in interactive computer games. High-end graphics boards, such as the NVIDIA 8800 GTX and their successors, featuring 500 G Flops and more, are now available for less than \$500, and their performance is

consistently growing at a triple of Moore's law that governs the growth of CPUs. Speedups of 1–2 orders of magnitude have been reported by many researchers when mapping CPU-based algorithms onto the GPU, in a wide variety of domains [10,18], including medical imaging [8,11,13,16]. These impressive gains originate in the highly parallel Same Instruction Multiple Data (SIMD) architecture of the GPU and its high-bandwidth memory access. For example, the NVIDIA 8800 GTX has 128 such SIMD pipelines while the most recent

* Corresponding author. Tel.: +1 631 632 1524.

E-mail address: mueller@cs.sunysb.edu (K. Mueller).

0169-2607/\$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.

doi:10.1016/j.cmpb.2009.09.003

NVIDIA card, the GTX 295, has 2×240 processors to yield a peak performance of 1.7 T Flops.

It is important to note, however, that the high speedup rates facilitated by GPUs do not come easy. They require one to carefully map the target algorithm from the single-threaded programming model of the CPU to the multi-threaded SIMD programming model of the GPU where each such thread is dedicated to computing one element of the (final or intermediate) result vector. Here, special attention must be paid to keep all of these pipelines busy. While there are 100s of SIMD processors on the GPU, many more threads need to be created to hide data fetch latencies. It is important to avoid both thread congestion (too many threads waiting for execution) and thread starvation (not enough threads available to hide latencies). These conditions are in addition to avoiding possible contingencies in local registers and caches that will limit the overall number of threads permitted to run simultaneously. For example, in [13], it was shown that a careful mapping of Feldkamp's filtered backprojection algorithm to the GPU yielded a $20\times$ speedup over an optimized CPU implementation, enabling cone-beam reconstructions of 512^3 volumes from 360 projections at a rate of 52 projections/s, greatly exceeding the data production rates of modern flat-panel X-ray scanners that have become popular in fully 3D medical imaging.

The compute-intensive nature of iterative reconstruction algorithms motivated their acceleration via commodity graphics hardware early on, first using graphics workstations [9] and later GPU boards [14]. We now refine these works by analyzing the acceleration of iterative reconstruction algorithms more closely in terms of the underlying GPU programming model and architecture. Iterative algorithms are different from analytical algorithms in that they require frequent synchronization which interrupts the stream of data, requires context switches, and limits the number of threads available for thread management. Iterative algorithms, such as Expectation Maximization (EM) [12] or the Simultaneous Iterative Reconstruction Technique (SIRT) [5] consist of three phases, executed in an iterative fashion: (1) projection of the object estimate, (2) correction factor computation (the updates), and (3) backprojection of the object estimate updates. Each phase requires a separate pass. Flexibility comes from the concept of *ordered subsets*, which have been originally devised mostly because they accelerated convergence. The projection data is divided into groups, the subsets, and the data within each of these groups undergo each of the three phases simultaneously. Here, it was found that the larger the number of subsets (that is, the smaller the groups) the faster is typically the convergence, but adversely also the higher the noise since there is more potential for over-correction. In EM, the method of Ordered Subsets (OS-EM) has become widely popular. OS-EM conceptually allows for any number of subsets, but the limit with respect to noise has been noted already in the original work by Hudson and Larkin [7]. For the algebraic scheme, embodied by SIRT, the Simultaneous Algebraic Reconstruction Technique (SART) [2] is also an OS scheme, but with each set only consisting of a single projection. In SART, the over-correction noise is kept low by scaling the updates by a relaxation factor $\lambda < 1$. Block-iterative schemes for alge-

braic techniques have been proposed as well [3]. In fact, the original ART [6] is the algorithm with the smallest subset size possible: a single data point (that is, ray or projection pixel).

It is well known that SART converges much faster than SIRT, and a well-chosen λ can overcome the problems with streak artifacts and reconstruction noise, allowing it produce good reconstruction results [1]. On the CPU, faster rate of convergence is directly related to faster time performance. But, as we previously demonstrated [15], when it comes to acceleration on a streaming architecture such as the GPU, SART is not the fastest algorithm in terms of time performance. In fact, the time performance is inversely related to the number subsets, making SIRT the faster scheme. This is due to the overhead incurred by the frequent context switching when repeatedly moving through the three iterative phases: projection, correction, and backprojection. In the same work, we also demonstrated that specific subset sizes can optimize both reconstruction quality and performance. However, the influence of the relaxation factor λ had not been considered in these experiments.

In the research reported here we now complete the study of [15] and also investigate the role of λ on GPU reconstruction speed performance, in relation to subset size. Here we find that an optimal choice of λ can have a great impact. Despite the fact that SART is slower than SIRT *per* iteration, an optimized λ -setting can reduce the number of required iterations for convergence to a greater extent than the extra per-iteration cost incurred by GPU data streaming and context switching. This reinstates SART and the lower subset versions of SIRT as the fastest iterative CT reconstruction schemes also on GPUs.

We shall note, however, that the optimal setting is likely application-dependent, which is not unique to the GPU setting alone. Here we make the reasonable assumption that a certain application will always incur similar types of data and thus an optimal parameter setting, once found, will likely be close to optimal for all data within that application setting. In that sense, our aim for this paper is not to provide optimal subset and relaxation factor settings for all types of data, but rather to raise awareness to this phenomenon and offer an explanation.

Our paper is structured as follows. First, in Section 2, we discuss iterative algorithms in the context of ordered subsets, present a generalization of SIRT to OS-SIRT, and describe their acceleration on the GPU. Then, in Section 3, we study the impacts of both subset size and relaxation factor on GPU reconstruction performance and present the results of our studies. Finally, Section 4 ends with conclusions.

2. Iterative reconstruction and its acceleration on GPUs

In the following discussion, we have only considered algebraic reconstruction algorithms (SART, SIRT), but our arguments and conclusions readily extend to Expectation Maximization (EM) algorithms as well since they are very similar with respect to their mapping to GPUs [14].

2.1. Iterative algebraic reconstruction: decomposition into subsets

Most iterative CT techniques use a projection operator to model the underlying image generation process at a certain viewing configuration (angle) φ . The result of this projection simulation is then compared to the acquired image obtained at the same viewing configuration. If scattering or diffraction effects are ignored, the modeling consists of tracing a straight ray r_i from each image element (pixel) and summing the contributions of the (reconstruction) volume elements (voxels) v_j . Here, the weight factor w_{ij} determines the contribution of a v_j to r_i and is given by the interpolation kernel used for sampling the volume. The projection operator is given as

$$r_i = \sum_{j=1}^N v_j w_{ij} \quad i = 1, 2, \dots, M \quad (1)$$

where M and N are the number of rays (one per pixel) and voxels, respectively. Since GPUs are heavily optimized for computing and less for memory bandwidth, computing the w_{ij} on the fly, via bilinear interpolation, is by far more efficient than storing the weights in memory. The correction update for projection-based algebraic methods is computed with the following equation:

$$v_j^{(k+1)} = v_j^{(k)} + \lambda \sum_{p_i \in OS_s} \frac{p_i - r_i}{\sum_{l=1}^N w_{il}} \quad r_i = \sum_{l=1}^N w_{il} v_l^{(k)} \quad (2)$$

For the purpose of this paper, we have written this equation as a generalization of the original SART and SIRT equations to support any number of subsets. Here, the p_i are the pixels in the M/S acquired images that form a specific (ordered) subset OS_s where $1 \leq s \leq S$ and S is the number of subsets. The factor λ is the relaxation factor, as mentioned above, which will be subject to optimization. The factor k is the iteration count, where k is incremented each time all M projections have been processed. In essence, all voxels v_j on the path of a ray r_i are updated (corrected) by the difference of the projection ray r_i and the acquired pixel p_i , where this correction factor is first normalized by the sum of weights encountered by the (back-projection) ray r_i . Since a number of backprojection rays will update a given v_j , these corrections need also be normalized by the sum of (correction) weights. For SIRT, these normalization weights are trivial.

2.2. GPU-accelerated reconstruction: threads and passes

As mentioned, the NVIDIA 8800 GTX board has 128 generalized SIMD processors. Up to very recently, the only way to interface with GPU hardware was via a suitable graphics API, such as OpenGL or DirectX, and using CG [4] (or GLSL or HLSL) for coding *shader programs* to be loaded and run on the SIMD *fragment processors*. With the introduction of a new API, Compute Unified Device Architecture (CUDA) [19], the GPU can now directly be perceived as a multi-processor. With CUDA, the CG fragments become the CUDA (SIMD) *computing threads*

and the shader programs become the *computing kernels*. With the CUDA specifications, much more information about the overall GPU architecture is now available, which helps programmers to fine-tune thread and memory management to optimize performance, viewing GPUs as the multi-processor architecture it really is. Reflecting this General Programming on GPUs (GPGPU) trend, new GPU platforms have now become available that do not even have graphics display capabilities, such as the NVIDIA Tesla board. Although we used GLSL shaders to obtain the results presented in this paper, similar symptoms also occur in CUDA where synchronization operations have to be formally called to finish executions of all threads within a thread block to resume the pipeline. After all, the underlying hardware and its architecture remain the same just the API is different.

The GPU memory model differentiates itself significantly from its CPU counterpart, posing greater restrictions on memory access operations in order to reduce latency and increase bandwidth. Here not only registers and local memory are reduced or even completely eliminated—the global memory also allows only read instructions during the computation. Further, the write operator can be executed only at the end of a computation, when the thread (or fragment) is released from the pipeline, to be blended with the target. Therefore, in general-purpose computing using GPUs, computations are triggered by initializing a “pass”. A pass includes setting up the computation region and attaching a kernel program to simultaneously apply specific operations on every thread generated. The data is then streamed into the pipeline, where the modification can be done only at the end of the pass. Cycles and loops within a program can be implemented either inside the kernel or by running multiple passes. The former is generally faster since evoking a rendering pass and storing intermediate results in memory are costly, but there exists a register count limit in the current hardware which prevents unconstrained use of loops in the kernel.

2.3. GPU-accelerated iterative reconstruction: implementation details

To illustrate the significance of our generalization of SART and SIRT into OS-SIRT, in context of GPU-accelerated computing, we shall now describe our specific GPU implementation in closer detail. As mentioned, iterative algorithms consist of three passes which are repeatedly executed until convergence: forward projection, corrective update computation, and backprojection. Of these, only the first and third pass are computationally challenging as they involve the reconstruction volume. We note that in SART each such pass involves only one projection, while in SIRT it involves the entire set of projections. In OS-SIRT the number of projections involved is determined by the size of the subset.

The high performance of GPUs stems from the parallel execution of a sufficiently large number of SIMD computing threads. Each such thread must have a target which eventually receives the outcome of the thread’s computation. In the forward projection the targets are the pixels on the projection images receiving the outcome of the ray integrations (Eq. (1)), while in the backprojection the targets are the reconstructed volume voxels receiving the corrective updates (Eq. (2)). Thus

the forward projection is inherently pixel-driven, while the backprojection is voxel-driven. These two mechanisms (or pipelines) are mainly distinguished by their interpolation scheme: the former interpolates (ray) samples in reconstruction volume space, while the latter interpolates (correction update) samples in projection image space. This constitutes an unmatched projector/backprojector pair, which has been successfully used in the past in other contexts [17].

In the work considered here, the 3D object estimate is represented as a three-dimensional texture contained in a bounding box of six polygons. The resolution of the texture is that of the volume to be reconstructed. The forward projector casts rays across this texture, and the backprojector updates the texture. Both observe the viewing geometry of the detector that is associated with the acquired data. We shall now describe the GPU implementations of these two pipelines more closely. For further detail the reader is referred to a number of available papers (see, for example [13,14]).

Forward projector: given the viewing geometry and position of the detector associated with the data, we first require for each of its rays the 3D coordinates of their entry and exit points into the volume texture's bounding box. These can be efficiently obtained with the GPU via OpenGL commands. We first transform the bounding box by the viewing geometry's transformation matrix and then render the bounding box polygons into the depth buffer. Here the viewport is chosen such that its resolution matches that of the projection data image. The entry points are computed by projecting the bounding box polygons and setting the GL depth buffer test to GL.LESS, which will only retain the depth coordinates of the front-facing polygons, and the exit points are computed by setting the GL depth buffer test to GL.GREATER, which will only retain the depth coordinates of the back-facing polygons. The ray direction vector for each ray (pixel) is then calculated by subtracting the ray's entry point 3D coordinates from the exit point 3D coordinates and normalizing this vector. Given a certain pre-defined ray step size (we use 1.0) each thread can then compute the number of sample points along its ray to initialize a volume traversal loop. Beginning at the volume entry point, it interpolates the volume texture (using the GPU's hardwired trilinear interpolation facility), adds this value to its ray integral (initialized to zero), and then advances the ray by adding the direction vector scaled by the step size to its current volume position. All rays advance in lock step and in parallel, each on a different parallel GPU processor. The result of this forward projection procedure is a texture of the same resolution than the input data.

Corrective update computation: this simple thread subtracts the texture obtained in the forward projection step from the input data texture and normalizes the result by a texture storing the sum of weights for each ray (computed in a preprocessing step with the volume texture set to unity). The result is stored in another 2D texture we call *corrective update texture*.

Backprojector: now the 3D volume texture becomes the rendering target. A thread is spawned for each voxel and the mapping from the voxel's 3D coordinate to the detector is determined via projective geometry calculations. Then the GPU's bilinear interpolation facility is called to compute the voxel update from the corrective update texture. The resulting value is then added to the 3D texture at that voxel position.

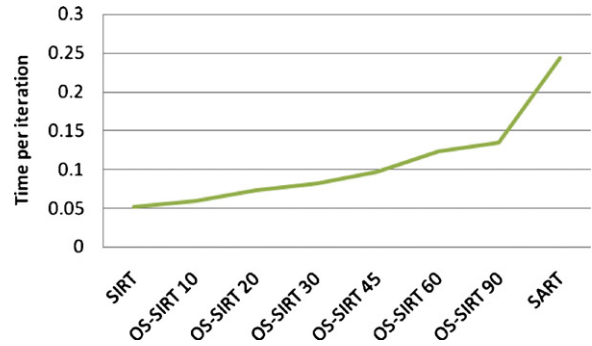


Fig. 1 – Time per iteration (in s) as a function of number of subsets.

We observe that since the forward projector uses a fairly good interpolation filter (trilinear) and a ray step size independent of the viewing angle, it produces a ray integral estimate that is a reasonably close approximation to an analytical integration (assuming ideal ray physics). The corrective updates of the backprojector, on the other hand, are not spread into the voxel grid via a trilinear function at regular intervals along the ray. Rather, the ray interval varies between 1 and $\sqrt{2}$ and the spreading only occurs in-slice via bilinear interpolation. Since the spreading (scattering) of thread results into multiple targets (the voxels in the trilinear neighborhood) is not supported on GPUs (and can also lead to hazards in parallel computation) this restriction is difficult to overcome. Fortunately, having a high-quality forward projector is of much higher importance in CT reconstruction since its integration result is compared to the integration result of the corresponding acquisition ray which then determines the corrective update.

Original	SIRT	OS SIRT 10
# iterations	87	47
time (s)	5.89	5.17
OS SIRT 20	OS SIRT 60	SART
32	15	6
11.28	9.9	10.28

Fig. 2 – Reconstructions obtained with the linear λ -selection schedule for various subset sizes for a fixed CC = 0.95 and 180 projections in an angular range of 180°. We observe that OS-SIRT with 10 subsets of 18 projections each reaches this fixed CC value 12% faster than SIRT and 50% faster than SART.

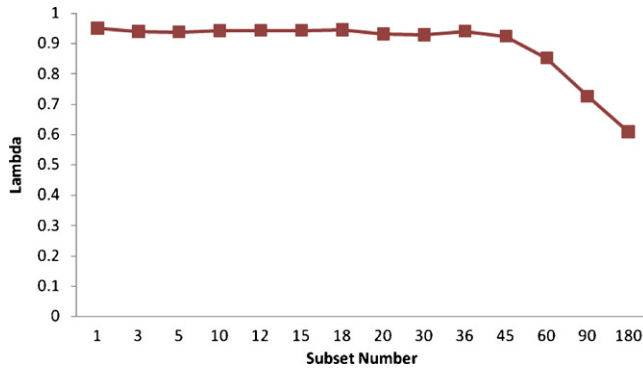


Fig. 3 – Optimal relaxation factor λ as a function of number of subsets for the optimal λ -selection scheme.

Original	SIRT	OS-SIRT 10
# iterations	95	10
time (s)	4.902	0.597
OS-SIRT 20	OS-SIRT 60	SART
5	2	1
0.367	0.246	0.244

Fig. 4 – Reconstructions with the optimized λ -selection scheme obtained with various subset sizes for a fixed CC = 0.95 and 180 projections in an angular range of 180° . We observe that now the number of subsets is directly related to wall-clock computation time, with SART being the fastest.

2.4. GPU-accelerated iterative reconstruction: extension to ordered subsets

From what has been discussed above we observe that ordered subsets (with the extreme case being SIRT) allow better control over the number of threads per kernel invocation. The more projections are grouped into a subset, the more pixel rays (and therefore threads) are spawned. This allows a better hiding of memory latencies since the threads waiting for memory can be (temporarily) replaced by other non-waiting threads not yet processed for the current program operation. An inherent limitation on the number of threads that can be managed this way is the available GPU (shared) memory. On the other hand, in the backprojection there are typically a sufficient number of threads since the number of voxels is an order of magnitude greater than the number of pixels in the projection images. However, backprojection threads are typically short

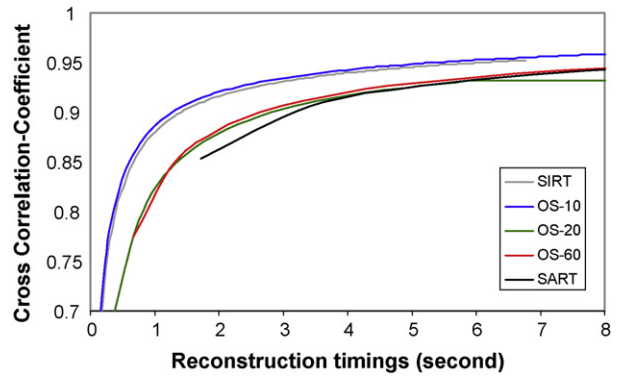


Fig. 5 – CC vs. wall-clock time for the linear λ -selection scheme. We observe that OS-SIRT achieves the reconstruction it in the smallest amount of time, within our GPU-accelerated framework.

which makes them less efficient. Since ordered subsets map each voxel to a set of projections this sequence of mappings increases the length of the kernel program and improves efficiency. A final advantage of ordered subsets is that they reduce the number of passes and the context switches that occur between the three repetitive phases of the iterative algorithm.

In the following we shall present our OS-SIRT algorithm in further detail. Eq. (2) above described the generalization of algebraic reconstruction into the OS configuration. What is left to define is how the subsets OS_s are composed and how λ is chosen for given number of subsets S . As specified above, OS_s is the set of projections contained in each subset, to be used in a pair of simultaneous forward projections and simultaneous backward projections. In our application, each subset has the same number of projections, that is $|OS_s| = |OS|$, which is typical. Thus, the total number of projections $M = |OS|S$. The traditional way of filling a certain subset OS_s is to select projections whose indices m ($1 \leq m \leq M$) satisfy $m \bmod S = s$. This is what has been adopted in OS-EM. In contrast, we use a randomized approach to fill the subsets, which we find yields better results than the regular subset population approach.

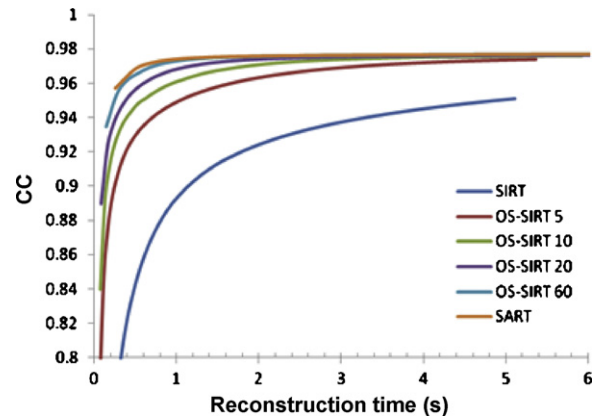


Fig. 6 – CC vs. wall-clock time for the optimized λ -selection scheme. We observe that there is now a clear ordering in terms of reconstruction time from small subsets to larger ones.

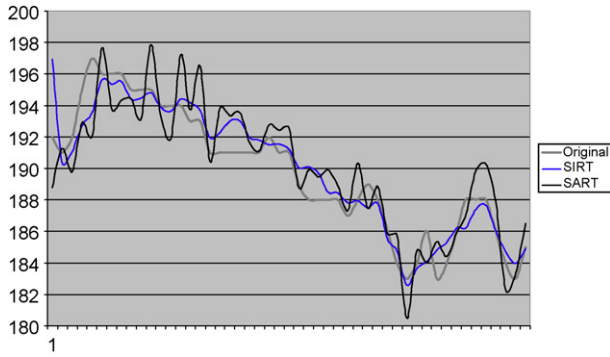


Fig. 7 – Line profiles of the image background for SART, SIRT, and the original Barbara image.

For this, we simply generate a projection index list in random order and sequentially divide this list into S subsets.

In [15] we proposed the following linear equation for the relaxation factor λ to be used for an arbitrary S , setting $\lambda = 1$ for SIRT and $\lambda = \lambda_{SART} = 0.1$ for SART:

$$\lambda = (\lambda_{SART} - 1) \left(\frac{S - 1}{N - 1} \right) + 1 \quad 1 \leq S \leq N \quad (3)$$

This scheme sought to balance the smoothing effect achieved by the application of a larger set of simultaneous projections (employed in SIRT) with that obtained by a lower relaxation factor (when a smaller set of projections is applied with SART or SIRT with smaller subsets). That is, the lesser projections in a subset, the lower the λ .

In our current work, we collected results for all possible integer-subsets, each for a representative set of λ -levels, and

used it to produce a more accurate $\lambda(S)$ function than the linear function in Eq. (3). In the following, we shall call the first scheme the *linear λ -selection scheme*, and the second method the *optimal λ -selection scheme*.

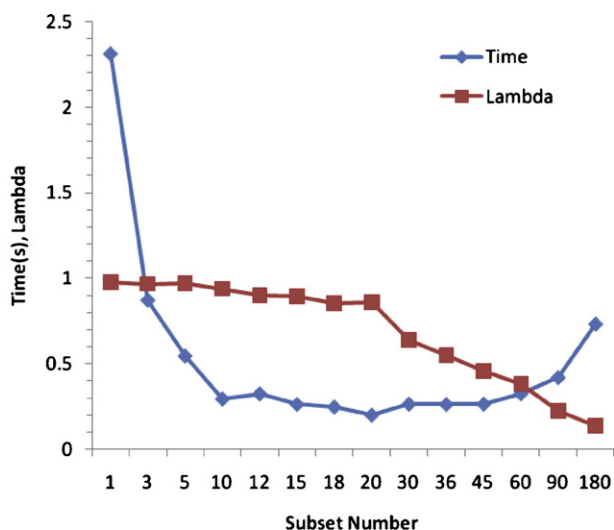
The arguments presented thus far as well as our prior observations in [15] justify the lower speed obtained with single iterations of SART or OS-SIRT with a large number of subsets, over single iterations with basic SIRT or OS-SIRT with a small number of subsets. The results we shall present below will show, however, that these extra costs incurred by SART and OS-SIRT with a large number of subsets are more than compensated by the faster convergence rates they can achieve, given the proper λ -settings obtained with a more informed λ -selection scheme.

3. Experiments and results

All our experiments were conducted on an NVIDIA 8800 GTX GPU, programmed with GLSL. For the first set of experiments we used the 2D *Barbara* test image to evaluate the performance of the different reconstruction schemes described above. We used this image, popular in the image processing literature, since it has several coherent regions with high-frequency detail, which present a well observable test for the fidelity of a given reconstruction scheme. The target 2D image is obtained by cropping the original image to an area of 256×256 pixels resolution. We obtained 180 projections at uniform angular spacing of $[-90^\circ, +90^\circ]$ in a parallel projection viewing geometry. We also simulated a limited-angle scenario, where iterative algorithms are often employed. Here, we produced 140 projections in the interval $[-70^\circ, +70^\circ]$. All reconstructions used linear interpolation.

Original	SIRT	OS-SIRT 3
# iterations / time (s)	100 / 1.88	85 / 1.77
OS-SIRT 10	OS-SIRT 60	SART
54 / 1.40	16 / 7.99	6 / 10.28

Fig. 8 – Reconstructed baby head using high-quality simulated projection data of the volume labeled ‘Original’. Results were obtained with the linear λ -selection scheme with various subset sizes for a fixed R-factor = 0.007 and 180 projections in an angular range of 180° . OS-SIRT with 10 subsets of 18 projections each reaches this set R-factor value 26% faster than SIRT and 86% faster than SART.



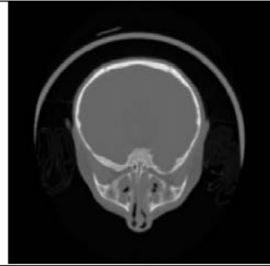
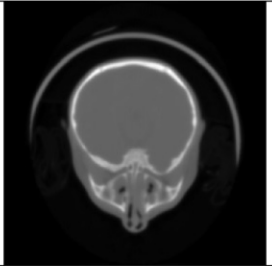
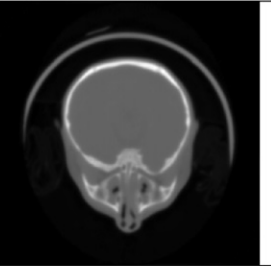
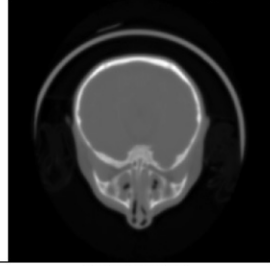
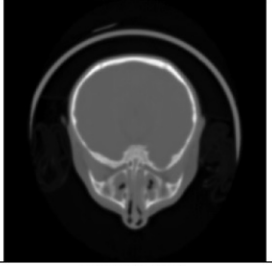
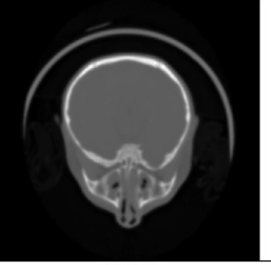
		
Original	SIRT	OS-SIRT 3
# iterations / time (s)	107 / 2.312	36 / 0.874
		
OS-SIRT 20	OS-SIRT 60	SART
6 / 0.203	6 / 0.328	6 / 0.734

Fig. 9 – Reconstruction results of the baby head obtained with the optimized λ -selection scheme with various subset sizes for a fixed R-factor = 0.007 and 180 projections in an angular range of 180°. Using the λ -selections shown above, OS-SIRT with 20 subsets of 9 projections each is the fastest in this case. It reaches this set R-factor value 91% faster than SIRT and 72% faster than SART. We also note that the time is about 7 times faster than with the linear selection scheme since the optimal λ -factor is higher.

We will use the cross-correlation coefficient (CC) as the metric to measure the degree of similarity between the original image and its reconstruction:

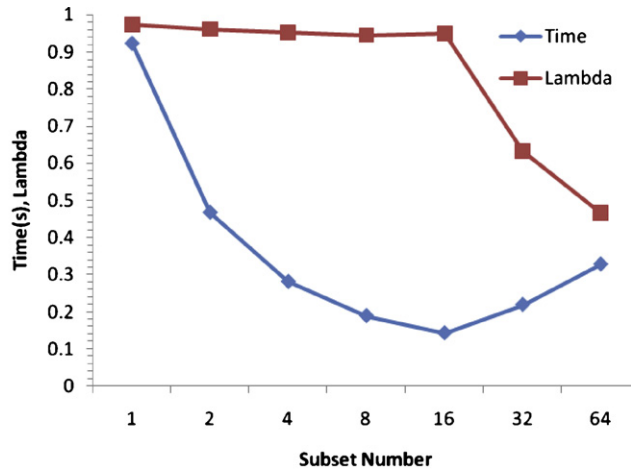
$$CC = \frac{\sum_j (r_j - \mu_r)(o_j - \mu_o)}{\sqrt{\sum_j (r_j - \mu_r)^2 \sum_j (o_j - \mu_o)^2}} \quad (4)$$

where j counts the number of image pixels, r_j and o_j are pixels in the reconstruction and original image, respectively, and the μ factors are their mean values.

We shall now explore if there is an optimum in terms of the number of subsets. Such an optimal subset size could

then be used to generate the best reconstruction in the smallest amount of (wall-clock) time. First, we evaluated the time per iteration for each configuration. Fig. 1 plots the time per iteration for each subset configuration, for the full-angle case of 180 projections. We observe a roughly linear relationship between the number of subsets and the time per iteration, with SIRT requiring the smallest and SART the longest time (about 5 times more than SIRT which is significant). This was to be expected given the arguments provided above.

However, in practice we are not interested in time per iteration but in time for convergence. Next, we test the two strategies presented (the linear and the optimal λ -selection



Original	SIRT	OS-SIRT 2
# iterations / time (s)	107 / 0.922	54 / 0.467
OS-SIRT 16	OS-SIRT 32	SART
8 / 0.143	7 / 0.219	7 / 0.328

Fig. 10 – Reconstruction of the baby head from a limited set of 64 projections in an angular range of 180° and the optimized λ -selection scheme shown above with various subset sizes for a fixed R-factor = 0.007. We observe that OS-SIRT with 16 subsets of 4 projections each is the fastest in this case. It reaches this set R-factor value 84% faster than SIRT and 56% faster than SART.

schemes) to choose the relaxation factor λ for each possible integer-subset configuration.

Fig. 2 shows the reconstruction results obtained with the linear λ -selection schemes, for a fixed CC (comparing reconstruction with the original) which means that all reconstructed images are nearly identical to each other (in terms of statistical error). We observe that the smaller the number of subsets, the greater the number of iterations that are required to reach the set convergence threshold. We measured that with the linear λ -selection scheme SART on the GPU takes nearly twice as long as SIRT and using 10 subsets (5 for the limited-angle case with fewer projections) achieves the best timing performance compared to the other subset configurations. For a CPU implementation, where the wall-clock time is directly related to the number of iterations, SIRT would be roughly $87/6 = 14$ times slower than SART. However, due to the

mentioned overhead involved in the GPU-based framework, different wall-clock times are produced with a GPU implementation and the ratio is not that severe.

We next evaluate the optimal λ -selection scheme. Fig. 3 shows the plot of the optimal λ found for each subset configuration for a $CC = 0.95$. We observe that the relationship is far from linear, with λ being relatively stable at around 0.95 until $OS = 45$ and then falling off to around 0.6 for SART. Thus, the linear curve underestimates λ most of the time, leading to unnecessarily small corrective updates. Fig. 4 shows the corresponding reconstruction results for the Barbara dataset. We see that SART now is the fastest algorithm, roughly 20 times faster than SIRT.

Figs. 5 and 6 compare the reconstruction quality vs. wall-clock time for both λ -schedules, respectively. We clearly observe that SIRT behaves very similarly, due to the simi-

lar λ -factor chosen, while the other subsets converge much more expediently for the optimal selection scheme, with SART reaching convergence after just one iteration, closely followed by OS-SIRT 60.

The tendency of SART to produce reconstructions noisier than the original and that of SIRT to produce reconstructions smoother than the original is also demonstrated in Fig. 7, where we show the renditions of a line profile across another area of the Barbara image (only for the original image and reconstructions with SART and SIRT).

We also studied a medical dataset, a slice of a baby head of size 256^2 . To assess the error we used the R-factor:

$$R = \frac{\sum_j (|SO_j| - |SC_j|)}{\sum_j |SO_j|} \quad (5)$$

where the SO and SC are the acquired and simulated sinograms, respectively. The R-factor is more practical for real reconstruction scenarios because it measures convergence based on the acquired data and not the (typically unavailable) object. All reconstructions were stopped once an R-factor of 0.007 was reached.

Fig. 8 shows the results obtained with the linear λ -selection framework. We observe that OS-SIRT with 10 subsets of 18 projections each reaches the preset R-factor 26% faster than SIRT and 86% faster than SART. Fig. 9 shows the reconstruction results of the baby head obtained with the optimized λ -selection scheme (shown above the table) from 180 projections in an angular range of 180° . We see that OS-SIRT with 20 subsets is fastest in this case. We also note that the time is about 7 times faster than with the linear selection scheme since the optimal λ -factor is higher. Finally, Fig. 10 presents reconstructions of the baby head from a limited set of 64 projections in an angular range of 180° again using the optimized λ -selection scheme. We observe that OS-SIRT with 16 subsets is the fastest in this case. The reconstruction time is close to a mere 10th of a second which demonstrates the capability of GPUs to also facilitate iterative reconstructions in interactive modes (10 frames/s).

4. Conclusions

We have shown that iterative reconstruction methods used in medical imaging, such as EM or SIRT, have special properties when it comes to their acceleration on GPUs. While splitting the data used within each iterative update into a larger number of smaller subsets has long been known to offer greater convergence and computation speed on the CPU, it can be vastly slower on the GPU. This is a direct consequence of the thread fill rate in the projection and backprojection phase. Larger subsets spawn a greater number of threads, which keeps the pipelines busier and also reduces the latencies incurred by a greater number of passes and context switches. This is different from the behavior on CPUs where this decomposition is less relevant in terms of computation overhead per iteration.

We have also shown that these effects can be mitigated by optimizing the relaxation factor λ , restoring the theoretical advantage of data decompositions into smaller ordered

subsets. In this paper we were mainly concerned with demonstrating that the poor per-iteration GPU performance for iterative CT with small subsets (in the limit SART) can be compensated for by choosing an appropriate λ -factor for the reconstruction. This leads to fast convergence of the small subset methods and thus provides a good amortization of the high per-iteration cost. Doing so in many cases allows even iterative reconstruction procedures to be run at near-interactive speeds, making them more applicable for mainstream CT reconstruction.

The question now remains how one chooses such an optimal λ -value in a practical setting. While this was not the topic of this paper, preliminary results have shown that the optimal number of subsets seems to vary depending on the domain application, the general reconstruction scenario, and also the level of noise present in the data. Thus, in order to identify the optimal subset number, as well as λ , for a new application setting and noise level, to be used later for repeated reconstructions within these scenarios, one may simply run a series of experiments with different numbers of subsets and λ -settings, and then use the setting combination with the shortest wall-clock time required for the desired reconstruction quality. In fact, such strategies are typical for GPU-accelerated general-purpose computing applications (GPGPUs). For example, the GPU bench was designed to run a vast benchmark suite [20] to determine the capabilities of the tested hardware.

We also believe that our findings with SIRT readily extend to EM since the two methods have, as far as the computations are concerned, similar operations and overhead, and we intend to study these dependencies more closely in future work. Finally, although we have used GLSL shaders to obtain the results presented in this paper, similar symptoms also occur in CUDA where synchronization operations have to be formally called to finish executions of all threads within a thread block to resume the pipeline. In general, the underlying hardware, its architecture, and the overall thread management remain the same—just the API is different, enabling a tighter control over the threads and also memory. In future work, we plan to study the reported effects to a more detailed extent in CUDA, to determine if a shift in the performance-optimal subset configuration occurs. But in fact, this is likely to happen for every new generation of the hardware.

Conflict of interest

None declared.

Acknowledgements

This work was partially funded by NSF grant CCR-0306438, NIH grant R21 EB004099-01 and GM31627 (DAA) and the Keck Foundation, and the Howard Hughes Medical Institute (DAA).

REFERENCES

- [1] A. Andersen, Algebraic reconstruction in CT from limited views, *IEEE Transactions on Medical Imaging* 8 (1989) 50–55.

- [2] A. Andersen, A. Kak, Simultaneous Algebraic Reconstruction Technique (SART): a superior implementation of the ART algorithm, *Ultrasonic Imaging* 6 (1984) 81–94.
- [3] Y. Censor, On block-iterative maximization, *Journal of Information and Optimization Science* 8 (1987) 275–291.
- [4] R. Fernando, M. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics*, Addison-Wesley Professional, 2003.
- [5] P. Gilbert, Iterative methods for the 3D reconstruction of an object from projections, *Journal of Theoretical Biology* 76 (1972) 105–117.
- [6] R. Gordon, R. Bender, G. Herman, Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography, *Journal of Theoretical Biology* 29 (1970) 471–481.
- [7] H. Hudson, R. Larkin, Accelerated image reconstruction using ordered subsets of projection data, *IEEE Transactions on Medical Imaging* 13 (1994) 601–609.
- [8] J. Kole, F. Beekman, Evaluation of accelerated iterative X-ray CT image reconstruction using floating point graphics hardware, *Physics in Medicine and Biology* 51 (2006) 875–889.
- [9] K. Mueller, R. Yagel, Rapid 3D cone-beam reconstruction with the Algebraic Reconstruction Technique (ART) by using texture mapping hardware, *IEEE Transactions on Medical Imaging* 19 (12) (2000) 1227–1237.
- [10] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, T. Purcell, A survey of general-purpose computation on graphics hardware, *Eurographics* (2005) 21–51.
- [11] T. Schiwietz, T. Chang, P. Speier, R. Westermann, MR image reconstruction using the GPU, *Proceedings of SPIE* 6142 (2006) 1279–1290.
- [12] L. Shepp, Y. Vardi, Maximum likelihood reconstruction for emission tomography, *IEEE Transactions on Medical Imaging* 1 (1982) 113–122.
- [13] F. Xu, K. Mueller, Real-time 3D computed tomographic reconstruction using commodity graphics hardware, *Physics in Medicine and Biology* 52 (2007) 3405–3419.
- [14] F. Xu, K. Mueller, Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware, *IEEE Transactions on Nuclear Science* 52 (2005) 654–663.
- [15] F. Xu, K. Mueller, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs, in: *MICCAI (Workshop on High-performance Medical Image Computing & Computer Aided Intervention)*, New York, September, 2008.
- [16] X. Xue, A. Cheryauka, D. Tubbs, Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: a simulation study, *Proceedings of SPIE* 6142 (2006) 1494–1501.
- [17] G. Zeng, G. Gullberg, Unmatched projector/backprojector pairs in an iterative reconstruction algorithm, *IEEE Transactions on Medical Imaging* 19 (5) (2000) 548–555.
- [18] <http://www.gpgpu.org>.
- [19] <http://www.nvidia.com/object/cuda.develop.html>.
- [20] <http://graphics.stanford.edu/projects/gpubench>.