©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.19.COIMG-167

# Sparse Data 3-D X-ray reconstructions on GPU processors

***Fernando Quivira, Simon Bedford, Richard Moore, John Beaty and David Castañón***

## Abstract

*The problem of obtaining 3-D tomographic images from geometries involving sparse sets of illuminators and detectors arises in applications like digital breast tomosynthesis, security inspection, non-destructive evaluation and other similar applications. In these applications, the acquired projection data is highly incomplete, so traditional reconstruction approaches such as filtered backprojection (FBP) lead to significant distortion and artifacts in the reconstruction. In this work, we describe an iterative reconstruction algorithm that exploits regularization to obtain well-posed inverse problems. However, the computations associated with these iterative algorithms are significantly greater than the FBP algorithms. We describe how we structure those computations to exploit GPU architectures to reduce the computation time of the iterative reconstruction algorithm. We illustrate the results on data computed from an experimental 3-D imaging system.*

## Introduction

The problem of obtaining 3-D tomographic images from geometries involving sparse sets of illuminators and detectors arises in many applications. In digital breast tomosynthesis [1], the desire to limit radiation exposure to levels comparable to mammography limits the collection geometry to a few angles. For non-destructive testing using x-ray imaging systems, these methods can potentially lead to more cost-effective hardware architectures and screening.

Since the acquired projection data is highly incomplete, traditional reconstruction approaches such as filtered backprojection [2] lead to significant distortion and artifacts in reconstruction. Instead, applications such as digital breast tomosynthesis [1] use model-based iterative reconstruction techniques ([3, 4] that control the occurrence of artifacts through combinations of accurate physics models and the use of suitable regularization techniques. However, these techniques have significantly higher computation requirements than transform-based techniques such as filtered-backprojection, which introduce significant delays in generating 3-D reconstructions for diagnosis or further exploitation. These delays introduce major limitations in the applicability of these systems for use in real-time systems that must maintain rapid imaging for timely inspection.

The need for acceleration of model-based iterative reconstruction methods was recognized early in tomosynthesis, and algorithms were designed for implementation in large processor clusters [6]. With the development of powerful graphics hardware such as Graphics Processing Units (GPUs), it became feasible to implement these iterative reconstruction algorithms in single computers [7, 8], leading to reported reductions in computation time comparable to those achieved in large clusters of processors.

In this paper, we describe a model-based iterative reconstruction algorithm that exploits regularization to reduce artifacts in sparse data CT based on the separable paraboloidal surrogates (SPS) algorithm [4]. This algorithm uses a physics-based model for transmission x-ray measurements based on Poisson statistics, which is useful in representing measurements at lower dosages. We describe how to structure the computations of this algorithm to map onto GPU architectures, and describe our implementation on an NVIDIA GPU system using CUDA. We demonstrate the efficiency of this mapping using data obtained from an experimental 3-D CT imaging system.

The rest of this paper is organized as follows: we describe the computations of the model-based iterative reconstruction algorithm in the next section. Subsequently, we overview the mapping of the algorithm computations onto GPU architectures. We describe experiments with reconstructions from projection data using an experimental sparse data CT tomography setup, comparing the GPU computation times with sequential reconstruction times. The last section discusses our conclusions and directions for future work.

## Model-based Iterative Reconstruction

Figure 1 illustrates the typical processing flow of iterative reconstruction algorithms. Starting from an initial guess of the reconstruction field, an iterative algorithm predicts the measurements using a physics-based model, computes the difference between the predicted measurements and the observed measurements, and uses these differences to compute updates to the estimated reconstruction field. These iterations are repeated until the algorithm stops. What changes among iterative reconstruction algorithms is how many measurements are processed and how are field updates computed.

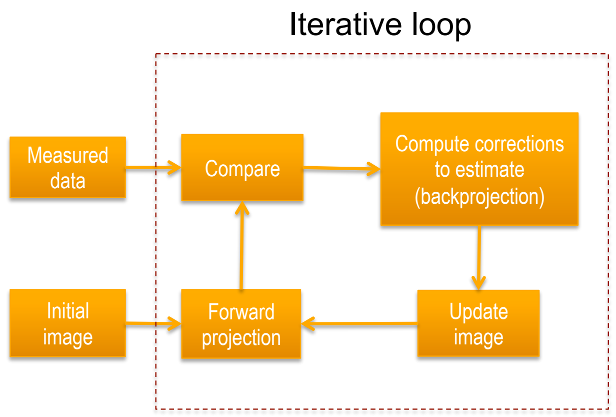

***Figure 1.*** *Typical processing flow for iterative reconstruction*

The general form of a model-based iterative reconstruction

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.19.COIMG-167

algorithm is derived from an optimization problem of the form

$$\arg\max_{\text{img}} \log P(\text{meas}|\text{img}) + f(\text{img})$$

where the first term is a data fidelity term that models the physics of the measurement system, and the second term is a regularization term that represents prior information and encourages the appearance of special structures in the reconstructed solution, such as smoothness. In our experimental algorithm in this paper, we are not using regularization to keep the description of the computation simple.

In our problem, let $\mu$ denote the discretized spatial map of effective linear attenuation coefficients, discretized over voxels indexed $j = 1,\ldots,P$. The tomographic projection measurements collected at the detectors are denoted by $Y_i, i = 1,\ldots,N$. We denote by $A$ the tomographic system matrix that relates the linear attenuation coefficients to the predicted intensities at the detectors $i = 1,\ldots,N$. This matrix is typically computed using the intersection lengths of the ray paths for measurement $i$ with the voxels [2]. The matrix $A$ has dimensions $N\times P$. Let $A_i$ denote the $i$-th row of $A$. We also denote by $r_i$ the background radiation intensity for measurement $Y_i$. We compute the $A_i$ rows using the approximation algorithm described in [5].

Let $b_i$ denote the source intensity for measurement $i$. The statistical model for the observation $Y_i$ collected at the detector is given by [4]:

$$Y_i \sim \text{Poisson}\{b_i e^{-A_i\mu} + r_i\}$$

Our data fidelity term is the negative log likelihood of the observations using this model. Using regularization, the resulting optimization problem is

$$\min_{\mu} \sum_{i=1}^{N} b_i e^{-\ell_i} + r_i - Y_i \log(b_i e^{-\ell_i} + r_i) + \frac{\beta}{2}\sum_{j=1}^{P}\sum_{k\in N_j} w_{jk}\psi(\mu_j - \mu_k)$$

where $\ell_i = A_i\mu$ is the forward projection of field $\mu$ to the measurements, $N_j$ is a neighborhood of voxel $j$, $w_{jk}$ are weights, and $\psi$ is a penalty function encouraging smoothness of the linear attenuation coefficients.

For the purposes of this study, we have chosen to model no background radiation (e.g. $r_i = 0$), and to use post-reconstruction smoothing rather than regularization, in order to concentrate on the difficult parts for acceleration of computation, which are the tomographic operations. The specific iterative reconstruction algorithm we have chosen for implementation is the separable paraboloidal surrogates (SPS) algorithm described in detail in [4], using ordered subsets. The algorithm partitions the measurements into $M$ disjoint subsets $S_m \in \{1,\ldots,N\}$, and processes the measurements one subset at a time to update the reconstructed field of linear attenuation coefficients. By partitioning the measurements into subsets, we reduce the needed memory in the GPU used to perform an update.

The basic organization of the algorithm computations is composed of major iterations, each of which is composed of minor iterations that process one subset of data at a time. During each minor iteration involving ordered subset $S_m$, the estimate of the field $\mu^{\text{old}}$ is updated with the data $\{Y_i, i\in S_m\}$.

The first step is to compute the forward projection of the current field estimate onto the detectors in the measurement subset $S_m$, as follows:

$$\begin{aligned}\hat{\mu} &= \mu^{\text{old}}\\ \ell_i &= A_i\hat{\mu}, \quad i\in S_m\\ \hat{y}_i &= b_i e^{-\ell_i}\end{aligned} \tag{1}$$

The next step is to back-project the parameters of each measurement in $S_m$ into the voxel field, as follows:

$$\begin{aligned} num(j) &= \sum_{i\in S_m} a_{ij}(\hat{y}_i - Y_i)\\ den(j) &= \sum_{i\in S_m} a_{ij}\ell_i\hat{y}_i\end{aligned} \tag{2}$$

The final step uses the back-projected statistics to update the field at each voxel, as

$$\mu_j^{\text{new}} = \left[\mu^{\text{old}}\left(1 + \frac{num(j)}{den(j)}\right)\right]_+$$

## GPU Implementation

For our implementation, we chose an workstation with an Intel Core i7 3970-X processor, and an Nvidia GeForce GTX Titan Black GPU. The Titan Black GPU has 2880 CUDA cores, with 6 Gigabytes of memory, and a single precision rating of 4.5 Teraflops. Figure 2 [9] is a diagram of the internal architecture of the Titan GPU, with 15 streaming multiprocessor (SMX) units, each of which is home to 192 cores. While this is not a top of the line GPU card, it has sufficient parallelization potential for our application.

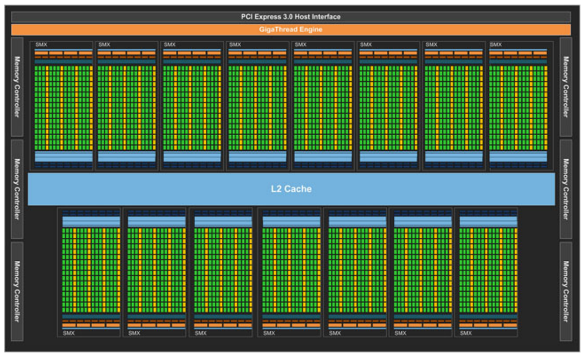

***Figure 2.*** *The GeForce GTX Titan GPU*

Figure 3 shows the contents of an SMX unit. In addition to the 192 single precision cores, the multiprocessor contains 64 double precision units, as well as 64 Kbytes of shared memory and 48 Kbytes of constant memory. We are not currently using any of these features in our implementation, as discussed below.

In order to understand the mapping of the algorithm onto the GPU architecture, below is an outline of the major computations of the ordered subset algorithm.

```
Initialize 3D volume field
```

DOI: 10.2352/ISSN.2470-1173.2016.19.COIMG-167

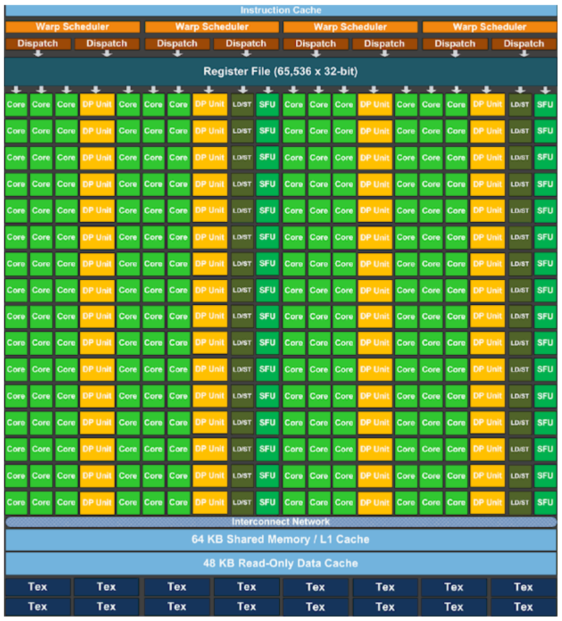
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

*Figure 3.* An SMX unit in the GeForce Titan

```
For number of iterations do
 For each subset S of measurements do:
   Copy ordered subset data to GPU
   For each projection in S do
     Compute coefficients of system matrix for
     projection
     Forward project volume field as in eq. (1)
   end For
   For each each projection in S do:
     Compute coefficients of system matrix for
     projection
     For each voxel j in 3D volume:
       Compute projection contribution to num(j),
       den(j) and accumulate as in eq. (2)
      end For
   end For
   For each voxel in field
      Update voxel field values as in eq. (3)
   end For
 end For
end For
```

An important motivation for the structure of our logic is to minimize memory access, and to keep most of the computations local to the threads being computed in each core of the GPU. As a consequence, we choose to compute the rows of the $A$ matrix at each iteration in each core, rather than storing the very large matrix. In addition to eliminating significant memory access time, this allows us to modify the illumination/detection geometry with very minor changes in the underlying software.

The first major step in our algorithm is based on partitioning the data by using ordered subsets. This allows us to divide the data into sizes that fit well into the GPU memory. We then perform forward projection, backward projection, and attenuation field update using the data from one ordered subset at at time. Thus, the first computation step is to copy the current ordered subset data into the GPU. This contains the measured intensities at each detector, as well as the source/detector geometry. We divide the measurements into blocks of 512 threads for the CUDA execution. Each thread computes the path of each ray in the forward projection phase: this involves computation of the elements of row $A_i$ individually by each thread, followed by accessing the current attenuation field values in the voxels of interest to predict the measurement observed at the detector as in eq. (1). This step is highly parallel, and easily distributed among the GPU cores.

Following the forward projection, we synchronize the GPU, and we again partition the measurements into blocks of 512 threads by source/detector combinations. The threads recompute the row elements $A_i$. This avoids the use of limited local memory for each GPU core. The recomputed row elements $A_i$ are used in the back projection step by each thread to compute the contribution to the numerator and denominator used in updating the field voxels that the ray path intersects, as in eq. (2). The contributions of all the threads are reduced to total update values per voxel, and stored in two voxel-sized arrays in GPU memory: the numerator and denominator arrays.

Once the back projection threads are completed, we use a second synchronization point to ensure that all the update information has been computed. We again partition the voxels into blocks of 512 threads to update the attenuation field in each voxel. This step is completely parallel, so each thread accesses the stored numerator and denominator values and updates the voxel value, as in eq. (2). After the update step, we introduce another GPU synchronization point before proceeding to load and process the next subset of measurements.

The main iterations continue across each of the ordered subsets of measurements, and across the major iterations. Given that our typical problem is in the order of $10^8$ measurements, there is plenty of work available to keep the 2880 computing cores busy, while avoiding memory access bottlenecks.

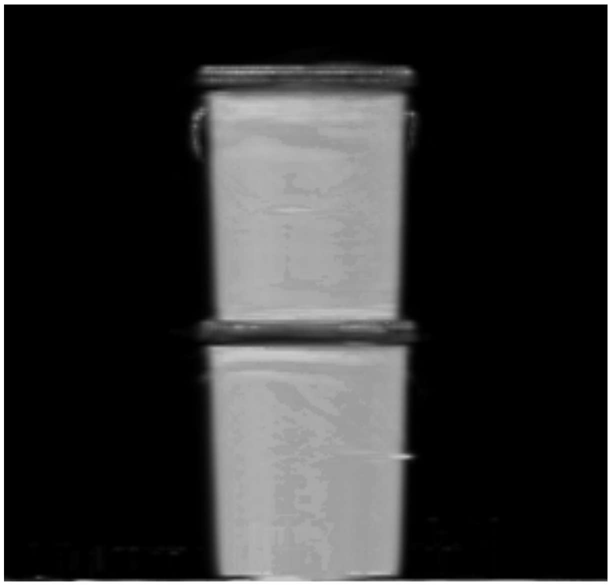

*Figure 4.* 2-D projection of reconstructed 3-D water bucket volume

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.19.COIMG-167

## Experiments

In this section, we describe the results of accelerating our reconstruction using three experimental data sets. The first experiment imaged 2 water buckets on top of each other. A 2-D projection of the reconstructed images for this experiment is shown in Figure 4. The buckets have thin metal handles, and are filled close to 90% up to the top, as shown in the reconstructions.

The second experiment focused on imaging a collection of 3 paint cans on a platform. A 2-D projection of the reconstructed scene is shown in Figure 5. The paint cans are nearly identical in size and content, but have different illumination because of the diverse placement in the scene.

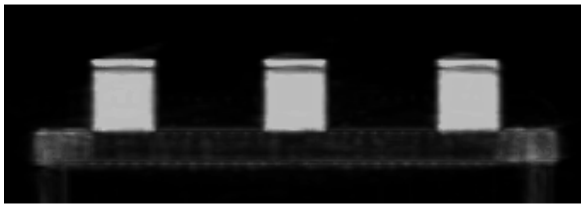

***Figure 5.*** *2-D projection of reconstructed 3-D paint cans volume*

The third experiment sought to image a collection of cylinders of different materials and heights, distributed over the imaging volume. A 2-D projection of the 3-D reconstruction of these cylinders is shown in Figure 6.

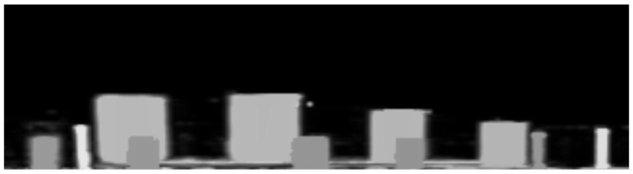

***Figure 6.*** *2-D projection of reconstructed 3-D collection of cylinders volume*

For each of these experiments, the reconstruction algorithm generated an estimate of the reconstructed attenuation field in a cubic region with dimensions 512 x 512 x 512 voxels. Storing this information in the GPU even at single precision takes a significant amount of memory. In terms of measurements, the specific experimental system we are using collects $10^8$ projections of this field, which we separate into four subsets of equal size. The algorithm performs 10 major iterations to obtain the final answer.

As described previously, our algorithms were implemented on an Intel Core i7 3970-X processor, with an Nvidia GeForce GTX Titan Black GPU. We ran the sequential version of the algorithm on the same processor, without using the GPU acceleration.

Table 1 contains the results of our experiments, comparing the computation time for the sequential implementation of our reconstruction code versus the GPU implementation. The times are reported as a fraction of the slowest sequential computation time. The times were computed as the average of 50 reconstruction times. Note that the sequential implementation of the algorithm is different: It avoids re-computation of the coefficients $A_i$ in the backprojection step, as it does not have the local memory limitations that the GPU cores have.

Our results consistently indicate an acceleration factor of near 60 from using the GPU implementation. The overall reconstruction time using the GPU architecture is approaching the requirements of what one would like to see for real-time imaging applications.

| | Sequential | Parallel |
|---|---|---|
| Paint Cans | 1.0000. | 0.0176 |
| Water Buckets | 0.9688 | 0.0162 |
| Cylinders | 0.8750 | 0.0164 |

**Table 1: Sequential and parallel computation times for different reconstructions. Time units are in fractions of slowest sequential computation time.**

Note that the current approach has taken a minimalist memory management approach to greatly increase parallelism. The rows $A_i$ are recomputed in the forward and backward projection iterations, and the main collaboration among the threads is the reduction of the information from the back projection operations to compute the numerator and denominator of the updates for the field intensities in the individual voxels.

## Discussion

In this paper, we describe an implementation of a model-based iterative reconstruction algorithm for sparse data computed tomography imaging on an NVIDIA GPU. The algorithm we chose to implement is the SPS algorithm described in [4]. We described the structure of the algorithm, and how it was mapped onto an Nvidia GeForce GTX Titan Black GPU architecture using CUDA.

We demonstrated the effectiveness of the GPU implementation when compared with a sequential implementation on the same architecture without using GPUs. For three sample test cases involving 3-D reconstructions with large numbers of voxels, the computation time was reduced by a factor of near 60 in each experiment.

Our GPU implementation approach was intended to make minimal use of local and shared memory, and used CUDA primitives for reduction. As such, it is suitable for a wide range of GPU sizes. There are a number of optimizations that we propose to explore in our future work to reduce further the computation time on GPUs by exploiting local and shared memory, as well as by using hierarchical reductions. We also propose to extend our implementation to include regularization as part of our voxel update procedure.

## Acknowledgments

The authors would like to acknowledge the support of Astrophysics, Inc. in this work.

## References

[1] R. G. Roth, A. D. A. Maidment, S. P. Weinstein, S. O. Roth, E. F. Conant, Digital Breast Tomosynthesis: Lessons Learned from Early Clinical Implementation, RadioGraphics, V. 34 .(2014).

[2] A. Kak, M. Slaney, Principles of Computerized Tomographic Imaging, IEEE Press.(1998).

[3] Z. Yu, J. Thibault, C.A. Bouman, K.D. Sauer, J. Hsieh, Fast model-based X-ray CT reconstruction using spatially nonhomogeneous ICD optimization, IEEE Trans Image Process, V. 20. (2011).

DOI: 10.2352/ISSN.2470-1173.2016.19.COIMG-167

[4] H. Erdogan, J. A. Fessler, Ordered subsets algorithms for transmission tomography, Phys. Med. Biol. No. 44. (1999).

[5] F. Jacobs, E. Sundermann, B. De Sutter, et al., A fast algorithm to calculate the exact radiological path through a pixel or voxel space, Journal of Computing and Information Technology, V. 6, No. 1 (1998).

[6] J. Zhang, W. Meleis, D. Kaeli, T. Wu, Acceleration of maximum likelihood estimation for tomosynthesis mammography, Int. Conf. Parallel and Distributed Systems. (2006).

[7] H. Yan, L. Ren, D. J Godfrey, F.-F. Yin, Accelerating reconstruction of reference digital tomosynthesis using graphics hardware, Medical Physics, V. 10. (2007).

[8] D. Schaa, B. Brown, B. Jang, P. Mistry, R. Dominguez, D. Kaeli, R. Moore, D. Kopans, GPU Acceleration of Iterative Digital Breast Tomosynthesis, GPU Computing Gems. (2011).

[9] T. Sandhu, Review: NVIDIA GeForce GTX TITAN 6GB graphics card overview, http://hexus.net/tech/reviews/graphics/51857-nvidia-geforce-gtx-titan-6gb-graphics-card-overview/, Feb. 19. (2013).

## Author Biography

*Fernando Quivira is a graduate student in the Electrical and Computer Engineering at Northeastern University, Boston, MA.*

*Simon Bedford is Director of Government Programs at Astrophysics Inc, in City of Industry, CA.*

*Richard Moore is Research Director for breast imaging at Massachusetts General Hospital, Boston, MA.*

*John Beaty is Director of Technology Programs for the Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems at Northeastern University, Boston, MA.*

*David Castañón is Professor of Electrical and Computer Engineering at Boston University, Boston, MA.*