Acceleration of Fluoro-CT Reconstruction for a Mobile C-Arm on GPU and FPGA Hardware: A Simulation Study

Xinwei Xue*[a], Arvi Cheryauka[b], and David Tubbs[b]

[a]School of Computing, University of Utah, 50 So. Central Campus Dr. Rm 3190, Salt Lake City, Utah 84112, USA;

[b]GE Healthcare, 384 Wright Brothers Drive, Salt Lake City, Utah 84116, USA

ABSTRACT

CT imaging in interventional and minimally-invasive surgery requires high-performance computing solutions that meet operational room demands, healthcare business requirements, and the constraints of a mobile C-arm system. The computational requirements of clinical procedures using CT-like data are increasing rapidly, mainly due to the need for rapid access to medical imagery during critical surgical procedures. The highly parallel nature of Radon transform and CT algorithms enables embedded computing solutions utilizing a parallel processing architecture to realize a significant gain of computational intensity with comparable hardware and program coding/testing expenses. In this paper, using a sample 2D and 3D CT problem, we explore the programming challenges and the potential benefits of embedded computing using commodity hardware components. The accuracy and performance results obtained on three computational platforms: a single CPU, a single GPU, and a solution based on FPGA technology have been analyzed. We have shown that hardware-accelerated CT image reconstruction can be achieved with similar levels of noise and clarity of feature when compared to program execution on a CPU, but gaining a performance increase at one or more orders of magnitude faster. 3D cone-beam or helical CT reconstruction and a variety of volumetric image processing applications will benefit from similar accelerations.

Keywords: CT, RECON, SIM, GPU / FPGA acceleration

1. INTRODUCTION

CT imaging in interventional and minimally-invasive surgery requires high-performance computing solutions that meet operational room demands, healthcare business requirements, and the constraints of a mobile C-arm system. The computational requirements of clinical procedures using CT-like data are increasing rapidly, mainly due to the need for rapid access to medical imagery during critical surgical procedures. Fast processing and display of the reconstructed image at any time before, during or after the procedure is critical. To achieve a higher spatial, contrast, or temporal resolution, the dimensions of input and output datasets used by the analytical reconstruction methods are increasing.[1] Use of iterative algorithms that offer advanced image processing features is also limited by performance of a single CPU computer.[2] On the other hand, the highly parallel nature of Radon transform and CT algorithms enables embedded computing solutions utilizing a parallel processing architecture to realize a significant gain of computational intensity with comparable hardware and program coding/testing expenses.[3]

In this paper, we describe a sample 2D and 3D CT problem and analyze the accuracy and performance results obtained on three computational platforms: a single CPU, a single Graphics Processing Unit (GPU), and a solution based on Field Programmable Gated Array (FPGA) technology. The purpose of this study is to explore the programming challenges and the potential benefits of embedded computing using commodity hardware components.

*Xinwei Xue: E-mail: xwxue@cs.utah.edu, Telephone: 1 801 536 4736

2. COMPUTER TOMOGRAPHY FUNDAMENTALS

The filtered-backprojection (FBP) techniques is the most popular analytical technique for 2D and 3D CT reconstruction.[4] It consists of two steps: ramp filtering and backprojection. Here we use 2D and 3D parallel beam CT reconstruction as examples to show the potentials of computation acceleration using state-of-the-art hardware.

2.1. Parallel Beam 2D Reconstruction

The 2D parallel beam FBP reconstruction process is as follows:

1. Filtering the projection

$$Q_\theta(n\tau) = \tau \sum_{k=0}^{N-1} h(n\tau - k\tau)P_\theta(k\tau), n = 0, 1, 2, ..., N-1 \tag{1}$$

where

Q_θ is the filtered projection at angle θ

h is the smoothed ramp filter

P_θ is the projection at angle θ

τ is the sampling interval

$n\tau$ is the n^{th} sample

N is the total number of samples in the projection.

2. Backprojection

$$f(x, y) = \frac{\pi}{N} \sum_{i=1}^{N} Q_\theta(xcos\theta_i + ysin\theta_i) \tag{2}$$

where f is the image under reconstruction, and (x, y) is the coordinate of one pixel.

The backprojection loop is the most computationally intensive part of the algorithm, thus it is our main target for acceleration.

2.2. Parallel Beam 3D Reconstruction

The backprojection step for 3D parallel beam reconstruction is similar to the 2D case. We use FBP technique for the parallel beam 3D reconstruction. In 3D parallel beam case, we use the same 2D reconstruction technique to reconstruct each slice.

3. CT ALGORITHM HARDWARE ACCELERATION: RELATED WORK

The intrinsic parallelism existing in the CT reconstruction computation draws attention for hardware acceleration of the performance for near real-time processing. There are a couple of potential solutions to this problem. The first is to design an application specific integrated circuit (ASIC)[5] for accelerating a particular algorithm. This is the most efficient way to accelerate an algorithm; however, ASICs are not flexible to adapt to the changes in the algorithm and are difficult to reuse in other applications. Moreover, designing and developing an ASIC incurs high development or non-recurring engineering costs. Another solution is to use the general purpose parallel computing platforms, such as multi-processor machines or distributed clusters of computers. This is the most flexible platform, but it suffers the high cost of a multi-processor supercomputer and the large volume of computer clusters, which is not appropriate for cost effective mobile C-Arm imaging.

Comparing to the two solutions mentioned above, the Field Programmable Gate Array (FPGA)[6] based computing platform provides a fairly good tradeoff between ASIC and PC cluster or supercomputer based parallel computing platforms. FPGA devices are configurable, and there are FPGA devices with specific DSP capabilities. It is fast, flexible, extensible, and easier than designing ASIC hardware. A disadvantage of FPGA-based systems is the large amount of time and effort required to redesign the computing architecture inside the FPGA for different algorithms. Also, it is difficult to manage the data flow in algorithms in which random data access occurs.

Another emerging hardware acceleration of parallelizable computations is to use the commodity hardware: the modern graphics cards, which have Graphics Processing Units (GPU) dedicated to parallel pipelined graphics processing. But, the applications of GPUs have gone far beyond the graphics applications, and found its way to a large variety of general purpose computing.[3, 7] Specifically for CT algorithm acceleration, the earliest attempt using graphics hardware dates back to the year 1994, when Calbral, Cam and Foran[8] utilized the texture mapping hardware for 3D reconstruction. With the introduction of modern GPUs in the last 5 years, both analytical (FDK) and iterative methods (SART, EM) have been implemented on graphics hardware.[9–11] GPUs have received lots of attention due to the following advantages:

GPU is powerful. Modern GPU can do computations with up to 32-bit floating-point precision. GPUs now have up to 512MB on-card memory with high memory bandwidth up to 54.4 GB/second. It has up to 24 pixel pipelines, which is essentially a SIMD (Single Instruction Multiple Data) parallel processor.

GPU is flexible and programmable. It has programmable vertex and fragment processors, which makes it possible for general purpose scientific computing (GPGPU), not limited to graphics applications.

GPU has rapid development cycle. The performance of GPUs is doubling every 6 months. GPU is inexpensive. Driven by multi-billion dollar video game industry, it costs less than \$500 for the latest card (e.g. Nvidia GeForce 7800 GTX).

GPU is scalable. Nvidia SLI and ATI Cross-fire technology makes it possible for multi-GPU solutions.

In this paper, we aim to explore the potential of two platforms in its application to the mobile C-Arm CT setting: The FPGA and GPU based platforms. We present the results with both platforms of a simple parallel beam 2D reconstruction example. Moreover, we experimented with the latest state-of-the-art GPUs and PCI Express interface, and analyzed their accuracy and performances. Finally we present the 3D parallel beam reconstruction using GPUs, which is readily extensible to cone-beam geometry and iterative CT reconstruction methods.

4. HARDWARE ARCHITECTURE & IMPLEMENTATION

4.1. FPGA and GPU Architecture

Modern high-end FPGAs integrate hard-wired processor cores, SRAM blocks, multipliers, digital clock managers and massive reconfigurable logic resources in a single chip. By combining these resources and customizing the computing architecture inside the FPGA, dedicated systems can be designed to accelerate different algorithms. The system architecture inside the FPGA can also be reconfigured to adapt to the changing demands of different algorithms, thus avoiding the need to design a totally new hardware. The high-level architectural diagrams of the Xilinx Virtex-2,[12] and Altera Stratix-2[13] are shown in Figure 1.

Commodity PC GPUs provide an adequate platform for general-purpose parallel scientific computing applications using the SIMD (Same Instruction Multiple Data) processing model. The modern GPU has 6 or more geometry pipelines and up to 24 pixel pipelines. The CPU/GPU structure is shown in Figure 2. Basically, the applications interact with GPU through OpenGL (or Direct3D) API commands. The data, represented as a set of vertices will flow through the OpenGL streaming rendering pipeline, and be written into framebuffer. The vertex and fragment processing is programmable. In vertex program (also called shader), the position and other geometry properties of vertices can be modified. The fragment shader can be considered as a computational kernel that will apply to the each screen fragments.[14, 15]

As summarized by Owens et. al.,[3] to program the GPUs: first, the programmer needs to determine the data-parallel portions of the application, which must be independent of each other. Second, draw a screen-size quad (that is same as the 2D computation size), so that there is one-to-one mapping between screen pixels (fragments) and the image voxels in computation. Third, the each fragments is processed by the active fragment

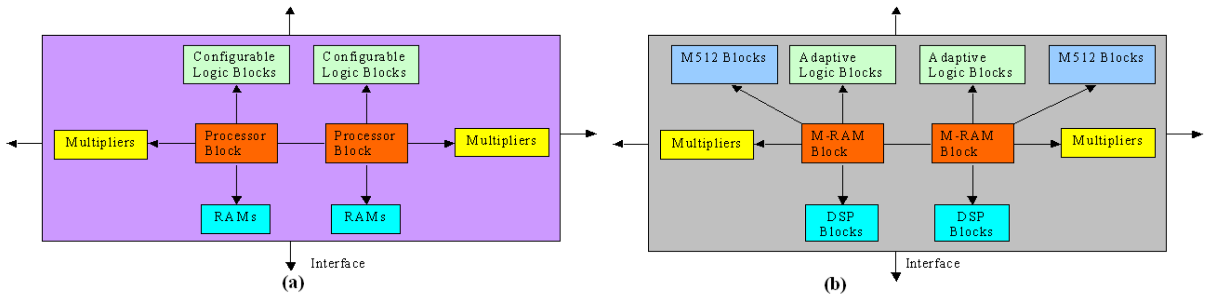

Figure 1. High-level FPGA Architecture. (a) Xilinx Virtex-2. (b) Altera Stratix-2

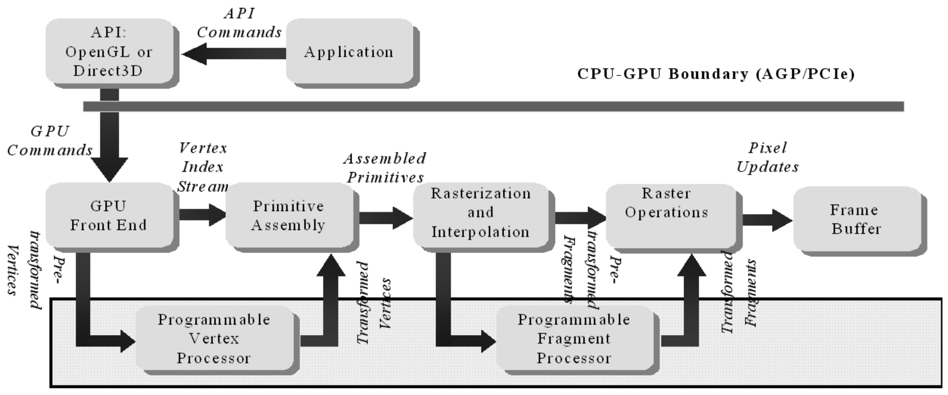

Figure 2. CPU/GPU Architecture

program. Fourth, the output, as an array of scalar or vector values, is written into framebuffer. The input data is represented in the texture (stream) format. Texture lookup can be done in fragment program.

4.2. Implementation

On the FPGA side, the 2D back-projection was performed using an FPGA solution based on dual Altera Stratix devices. The calculations were performed by reading data from one SDRAM (Synchronous Dynamic Random Access Memory) interface and writing into an alternating SDRAM interface to yield maximum bandwidth. The multiply/accumulate operations were 32-bit fixed point. SDRAM clock rate, capacity of SDRAM interface, and number of interfaces available have been identified as critical factors to increasing performance.

On the GPU side, there are two approaches to implement the backprojection loop. One is the pixel-driven approach, using the texture spreading technique proposed by Cabral et. al.,[8] where the projections are rotated and accumulated onto the image to be reconstructed. There are several disadvantages to this method. First, the reconstructed image is usually much smaller than the projection size (e.g. reconstruct a 256x256 image from projection size of 512, 768 or larger). Thus, the projections, if not pre-scaled to the same size of the reconstruction image, will result in poor performance, due to wasted operations. Scaling would cause interpolation inaccuracy. Another disadvantage is the precision limitation of hardware rotation operation on textures and the accumulation buffer. Finally, this forward transform (instead of inverse transform, mapping the target image to the source image) would generate artifacts.

The other approach is the voxel-driven approach, where the back-projection is done on the GPU using the GPU's fragment shader, in a similar fashion as one would implement the algorithm in plain C/C++ on a CPU, where the inverse transform is used. Render-to-texture (RTT) techniques allows us to draw the result into an offline buffer (pbuffer), which does not require display. Furthermore, it enables computations with up to 32-bit floating point precision. We conducted our GPU experiments using this approach. In our implementation, we utilized the ping-pong pbuffer technique to reduce the context switches.

5. RESULTS

5.1. Experimental Settings

For parallel beam 2D reconstruction, the projection dataset of 165 equiangular views and the detector bin of 512 equidistant pixels were used to acquire the data from a 2D Shepp-Logan phantom. The sinogram has been contaminated by electronic noise of PSF (Point Spread Function)=0.1, Gaussian multiplicative noise of 2% magnitude, and then convolved with a ramp-filter and formatted into floating point and 16-bit integer data. The image to reconstruct is of size 256 square. For the parallel beam 3D case, we reconstruct 256 cubed volume from 165 views, each projection has a image size of 512x512.

The back-projection loop has been implemented in Matlab, C++, using the CG (C for Graphics) programming language on the GPU, VHDL for the FPGA devices and executed on the CPU, GPU, and FPGA chips, respectively. The PC we used has a 3.4GHz Pentium 4 processor and 2GB memory. The capabilities of three graphics cards have been explored: The ATI Radeon X700 Pro, with 256MB of memory and 8 pixel pipelines, the Nvidia GeForce 6800 Ultra with 256MB memory and 16 pixel pipelines, and the GeForce 7800 GTX with 256MB memory and 24 pixel pipelines. All these graphics cards have PCI-Express X16 interface.

5.2. Results: 2D Case

The 2D reconstruction results of a 256x256 image compared to the 256x256 discretized phantom show very small differences (Figure 3). The profiles shown in the figure are row 128 and column 128. RMS (Root Mean Square) errors, computational times and speed up gains are summarized in Figure 4.

The timing is conducted over the average of a number of 10 computations. The timing does not include the time transferring data between GPU and CPU (only once for some problem size). We can compute 4-5 views in each fragment program(per render pass), thus reduces loop overhead. The accumulation of back-projection values has been performed in single-precision floating point on the CPU and GPU, and using 32-bit integer format on the FPGA. The difference between 16-bit integer input and 32-bit float input to GPU is that, the GPU texture mapping hardware can do hardware interpolation with 16-bit integer input when doing texture lookup (hence reduce the length of the fragment shader), while the float interpolation can only be done in fragment shader. The accumulation is done in floating point precision for each case. Since ATI GPUs use 24-bit float precision internally, it is less accurate than the Nvidia GPUs, which uses 32bit float internally. Another thought derived from Figure 4 is that, the result with 16 bit input on GPUs already achieves enough accuracy (small RMS difference compared against 32-bit input). With equivalent precision, the GPU and FPGA achieve the equivalent accuracy as the CPU. We realized a speedup of 44X on the Nvidia GeForce 7800 GTX GPU (relative to the C/C++ implementation). The performance gain realized on the FPGA hardware exceeded 24X.

5.3. Results: 3D Case

The 3D parallel beam reconstruction is implemented on CPU and GPU (Nvidia GeForce 6800 Ultra and GeForce 7800 GTX). The results are shown in Figures 5 and 6. With 16-bit integer input, we achieved a speedup of 71X.

As a summary, both hardware acceleration approaches can achieve much faster results than single CPU implementation. Also, the GPU approach outperforms the FPGA platform for ray-projection type of applications. Due to the limitation of the FPGA's on-chip memory and the bus speed, the GPU acceleration appears to be a more preferable choice of lower cost and better performance.

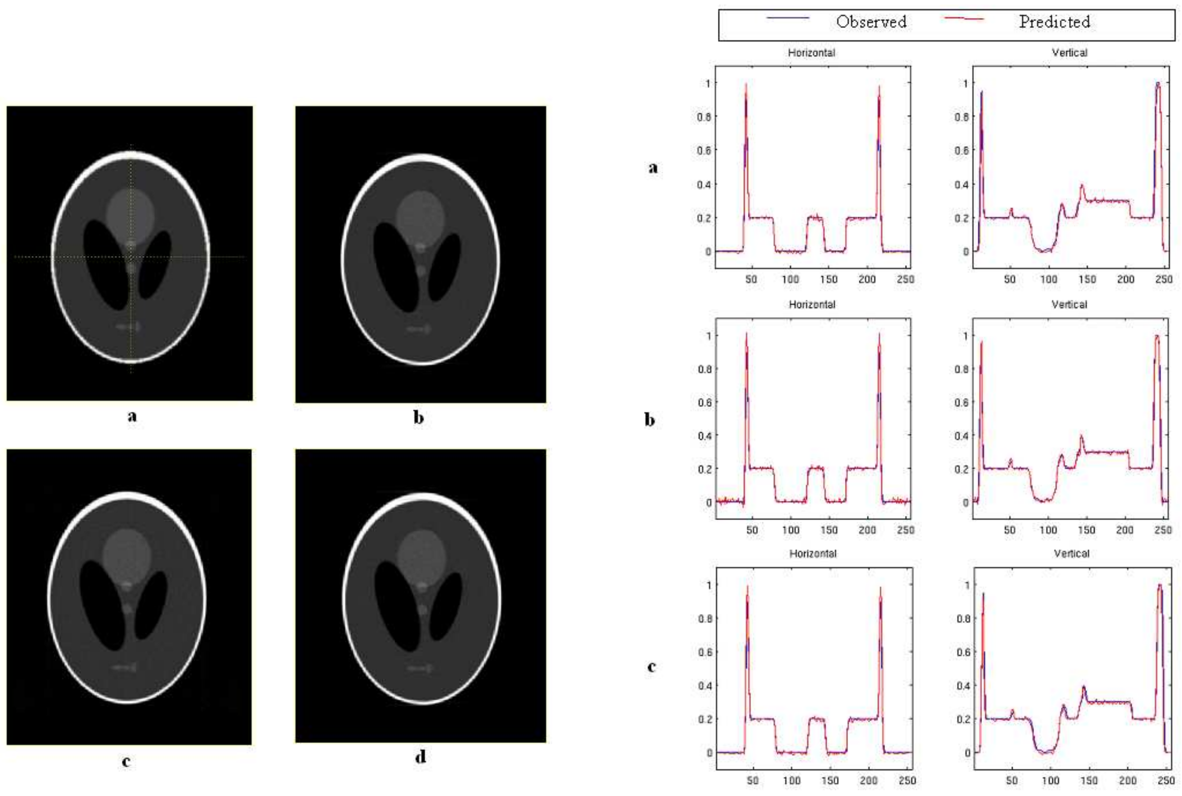

Figure 3. Parallel Beam 2D Reconstruction Results. Left Two Columns: Images (a) Shepp-Logan Phantom (b) CPU Recon (c) GPU Recon (d) FPGA Recon. Right Two Columns: Profile Comparison: horizontal profile and vertical profile (each row). (a) CPU Recon (b) GPU Recon (c) FPGA Recon. The blue line is phantom, and the red line is the reconstruction result.

	RMS	Time (ms)	Speedup
CPU (C++) 32-bit float	0.03040	507	1
FPGA (Altera) 32-bit integer	0.03042	22	23.1
ATI X700 Pro **GPU** 32-bit float input	0.03195	45.3	11
ATI X700 Pro **GPU** 16-bit input	0.03200	36	14.1
Nvidia GF6800 **GPU** 32-bit input	0.03042	36	14.1
Nvidia GF6800 **GPU** 16-bit input	0.03100	21	24
Nvidia GF7800 **GPU** 32-bit input	0.03042	23.8	21
Nvidia GF7800 **GPU** 16-bit input	0.03100	11.5	44

Figure 4. Parallel Beam 2D Reconstruction Results: Speedup and Accuracy

6. CONCLUSIONS

In this paper, we have implemented the back-projection loop using the same acquisition geometry setup and the same set of filtered data on the CPU, GPU, and FPGA platforms. We have shown that hardware-accelerated 2D and 3D CT image reconstruction can be achieved with similar levels of noise and clarity of feature when compared to program execution on a CPU, but gaining a performance at one or more orders of magnitude faster.

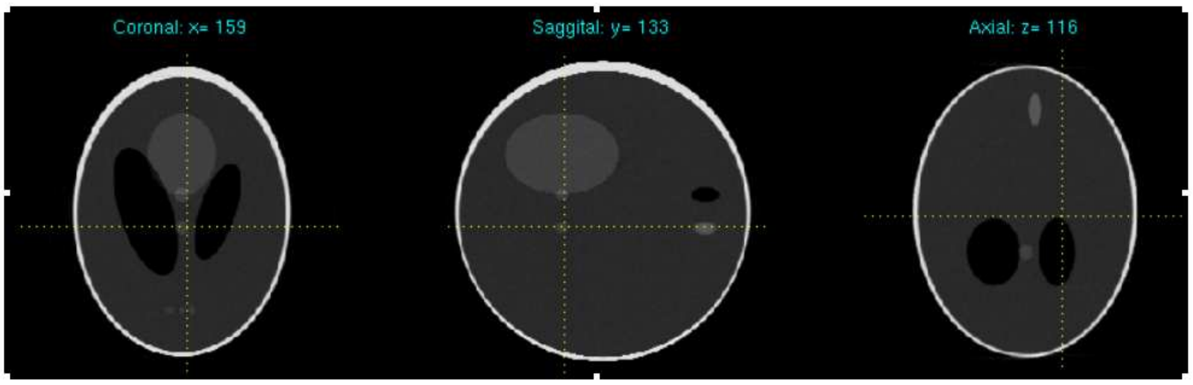

Figure 5. Parallel Beam 3D Reconstruction Results: Coronal, Sagittal and Axial Slices.

	Time (s)	Speedup
CPU (C++)	84	1
Nvidia GF6800 Ultra GPU (32bit in)	2.8	30
Nvidia GF6800 Ultra GPU GPU (16bit in)	1.8	46.7
Nvidia GF7800 GTX GPU GPU (32bit in)	1.84	45.7
Nvidia GF7800 GTX GPU GPU (16bit in)	1.18	71

Figure 6. Parallel Beam 3D Reconstruction Results: Computation Time and Speedup

3D cone-beam and helical CT reconstruction and a variety of 2D or volumetric image processing applications will also benefit from similar accelerations. At the same time, very large-scale computing on a stand-alone PC may cause time delays transferring data between graphics and main memory, hence the faster CPU/GPU interface, such as PCI-Express bus will be an immediate relief. Further optimization of memory management and dataflow arrangement techniques seem to be the next logical step. Moreover, utilizing the more advanced hardware features that becomes available lately and in the future, would further enhance the computation performance.

ACKNOWLEDGMENTS

We are very thankful to Gordon Pope (GE Healthcare OEC) for the implementation of the 2D backprojection on FPGA platform. We are thankful to Stephen Zingelewicz (GE Global Research) and Dmitry Samsonov (GE Healthcare, Haifa) for the helpful discussions on GPUs. We also thank the support from the Image Quality Group and Systems Engineering Department, GE Healthcare OEC.

REFERENCES

1. R. Clackdoyle, “Fully 3d reconstruction theory in perspective,” in *Proceedings of Fully 3D Reconstruction in Radiology and Nuclear Medicine*, 2005.
2. J. Nuyts, B. D. Man, P. Dupont, M. Defrise, P. Suetens, and L. Mortelmans, “Iterative reconstruction for helical ct: a simulation study,” *Phys. Med. Biol.* **43**, pp. 729–737, 1998.
3. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” in *Eurographics 2005, State of the Art Reports*, pp. 21–51, Aug. 2005.
4. A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, IEEE Press, 1988.
5. *TeraRecon Inc. http://www.terarecon.com/home.html* .

6. J. Li and C. Papachristou, “An fpga-based computing platform for real-time 3d medical imaging and its application to cone-beam ct reconstruction,” *The Journal of Imaging Science and Technology* **49**(3), pp. 237–245, 2005.
7. *GPGPU. http://www.gpgpu.org/* .
8. B. Cabral, N. Cam, and J. Foran, “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware,” in *Symp. On Volume Visualization*, pp. 91–98, 1994.
9. F. Xu and K. Mueller, “Near-interactive cone-beam computed tomography on comodity pc graphics hardware,” in *Proc. IEEE Medical Imaging Conference'03*, 2003.
10. F. Xu and K. Mueller, “Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware,” *IEEE Transaction of Nuclear Science* , 2005.
11. *Mercury Computer Systems. http://www.mc.com/* .
12. *Xilinx Virtex II Devices. http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas* .
13. *Altera Stratix II Devices. http://www.altera.com/products/devices/stratix2/st2-index.jsp* .
14. M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*, third edition ed., 1999.
15. R. Fernando, ed., *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Addison-Wesley, 2004.