Bio-Medical Materials and Engineering 26 (2015) S1027–S1035
DOI 10.3233/BME-151398
IOS Press

A block-wise approximate parallel implementation for ART algorithm on CUDA-enabled GPU

Zhongyin Fan[a, b] and Yaoqin Xie[a,*]
[a]*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China*
[b]*Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, China*

Abstract. Computed tomography (CT) has been widely used to acquire volumetric anatomical information in the diagnosis and treatment of illnesses in many clinics. However, the ART algorithm for reconstruction from under-sampled and noisy projection is still time-consuming. It is the goal of our work to improve a block-wise approximate parallel implementation for the ART algorithm on CUDA-enabled GPU to make the ART algorithm applicable to the clinical environment. The resulting method has several compelling features: (1) the rays are allotted into blocks, making the rays in the same block parallel; (2) GPU implementation caters to the actual industrial and medical application demand. We test the algorithm on a digital shepp-logan phantom, and the results indicate that our method is more efficient than the existing CPU implementation. The high computation efficiency achieved in our algorithm makes it possible for clinicians to obtain real-time 3D images.

Keywords: CT, ART, GPU, block-wise, approximate parallel

1. Introduction

CT is often used to acquire volumetric anatomical information in clinical diagnosis and treatment. However, the excessive x-ray imaging dose from frequent scans is a potential concern. However, it can be greatly reduced by decreasing the number of x-ray projections.

The filtered back-projection (FBP) [1] algorithm remains the most widely used reconstruction method in CT. When the number of projections is insufficient, artifacts have a great influence on image quality and diagnosis rate; this is an inherent defect of the FBP method. As alternatives, iterative algorithms, such as Algebraic Reconstruction Technique (ART) [2], Simultaneous Algebraic Reconstruction Technique (SART) [3], Simultaneous Iterative Reconstruction Technique (SIRT) [4], Multiplicative Algebraic Reconstruction Technique (MART) [5], Maximum-Likelihood Expectation-Maximization (MLEM) [6], Ordered-Subsets Expectation-Maximization (OSEM) [7], result in less image noise, but have a longer reconstruction time. It has been shown that less than ten iterations are

[*] Address for correspondence: Yaoqin Xie, Research Centre for Medical Robotics and Minimally Invasive Surgical Devices, Institute of Biomedical and Health Engineering, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China. Tel.: 86-755-86392281/13620902469; Fax: 86-755-86392299; Email: yq.xie@siat.ac.cn.

sufficient to reconstruct a low-contrast, 3-D object. Compared with the FBP method, the required number of projections in iterative algorithms is typically smaller [8, 9].

However, most implementations of iterative algorithms are time-consuming, which is unacceptable during many medical procedures, particularly those which are invasive. For example, more than 2.5 hours are required to reconstruct a 128^3 volume from 80 projections with the ART method on a modern workstation [10]. Therefore, the necessity of a real-time iterative algorithm is evident. In recent years, more researchers have paid close attention to the highly parallel computing power of a Graphics Processing Unit (GPU) for general purpose computations. The GPU is a competitive platform for solving parallel problems. Many issues in medical physics can be formulated as data-parallel tasks that use the GPU to reduce computing time [11].

Iterative reconstruction is computationally challenging, previously viewed as a critical target for GPU acceleration. Whereas analytical reconstruction is parallel, iterative reconstruction is fundamentally sequential. Hence, algorithms that perform minimal computation within each iteration do not perform as well as they could, since the GPU is not parallel. For instance, ART is not suitable for the GPU because each iteration only processes a single projection line.

In this paper, we propose a novel block-wise approximate parallel implementation of ART on Computer Unified Device Architecture (CUDA)-enabled GPU. We allot the rays into blocks, the rays in the same block are computed parallel. The resulting method has another important feature in that GPU implementation is going to meet the industrial and medical demands.

This paper is organized as follows. The ART algorithm is presented in section 2. The proposed algorithm will be discussed in detail in section 3. Computer implementation issues and the results will be analyzed in section 4. Computer simulated data will be used to compare the proposed methods. Issues with the data will also be discussed. Section 5 provides the summary and the plans for future work.

2. Background: ART algorithm

Typically, a CT system could be modeled by the following equation:

$$\begin{gathered} Wf = P \\ p^i = \sum_{j=1}^{N} w_{ij} f_j, i = 1, 2, ..., M \end{gathered} \tag{1}$$

where the observed data is denoted by $p=[p_1, p_2,.., p_M]^T$, original image is $f=[f_1, f_2,.., f_N]^T$, and $W=(w_{ij})$ is a non-zero M*N matrix, where w_{ij} is the length of the intersection of the i th ray with the j th pixel, M is the total number of rays and N is the number of cells.

The equations correspond to rays from the x-ray source, through the volume, to the detector pixels, i.e. the line integral of the linear attenuation coefficient. We consider the problem of reconstructing the image f from the data p.

The ART algorithm, based on Kaczmarz's method [12] for solving linear systems of equations $Ax=b$ and first proposed by Gordon et al. [2], is a tomographic reconstruction method to solve the problem of K dimensional reconstruction from several ($K-1$) dimensional projections in electron microscopy and radiology.

More formally, the ART correction for voxel j, to be performed for each correction step k, is written as follows [2]:

$$f_j^{(k+1)} = f_j^{(k)} + \lambda \frac{p^i - \sum_{n=1}^{N} w_{in} f_n^{(k)}}{\sum_{n=1}^{N} w_{in}^2} w_{ij} \tag{2}$$

Here, λ is a relaxation factor. Often, a positive constraint is applied to the voxels based on the assumption that negative attenuation values are not possible.

3. Method

The ART algorithm can be simplified as follows:

```
Algorithm 1:ART algorithm
1.Start by setting $f_0$
2. For each iteration k
    2.1 : For each angle $\theta$
        2.1.1 : Forward projection, for each ray calculating the correction
            $correction = p^i - \sum_{n=1}^{N} w_{in} f_n^{(k)}$
        2.1.2 : back projection, for each pixel j in the ray
            $f_j^{(k+1)} = f_j^{(k)} + \lambda \frac{w_{ij}}{\sum_{n=1}^{N} w_{in}^2} correction$
        2.1.3 : Repeat steps 2.1.1 - 2.1.2 for all the rays in angle $\theta$
    2.2 : Repeat steps 2.1.1 - 2.1.3 for all angles
3.iteration will be in progress until convergence
```

This process is our first choice, but it can't be ideally mapped to the GPU hardware.

More often than not, each ray line from the source to the detector unit can be computed with each GPU thread directly. However, if we were to perform this forward-projection and backward-projection operation by the ray-driven algorithm in the GPU reconstruction implementation with each thread in charge of updating value of voxels along a ray line, a memory conflict problem would occur. This issue is due to the possibility that different GPU threads may update the same voxel value synchronously. As a result, we must wait until one thread finishes updating before we can update another, severely limiting the exploitation of GPU's massive parallel computing power.

For each ray line, the ray line goes through a small number of volume voxels, since W is a sparse matrix. In this paper, we assign the sequence for the rays to be performed, allotting rays into blocks. This means that the rays in the same block are computed parallel. We should make sure the rays in the same block do not have the possibility of updating the same voxel value by different GPU threads.

In order to explain why the rays cannot be computed parallel, how to allot the rays, and why the rays in the same block can be computed parallel, we use a 2D fan beam as an example. Then we extend to a 3D CBCT, another modality which can also be similar.

3.1. 2D fan beam case

In Figure 1, the X-ray source moves in an ideal circle within the centre of the objection, (*u-1*), *u* and (*u+1*) are three units in the detector. L_1, L_2, L_3 represent three rays from the source to the detector unit. The three rays all go through the pixel $f_{i,j}$, and thus cannot be computed parallel.

As described in Figure 2, if we try to make the rays sparse enough, no two of the rays would go through the same pixel, we can compute the rays parallel. Thus, we put rays into a subset, whose step is large enough. In this means, we allot the detector units into blocks. The detector units in a block are sparse enough to make sure that each set of rays goes through a different pixel.

Let us assume that each projection view has *N* number of rays. *S* represents a set of total number of rays presented in a projection view, and *M* is the number of blocks and the step to choose, such that:

$$S = \{S_1, S_2, ..., S_M\} \tag{3}$$

where S_1={1, *M*+1, 2**M*+1,...,*N*-*M*+1}, S_2={2, *M*+2, 2**M*+2,...,*N*-*M*+2},.., S_i={i, *M*+i, 2**M*+i,...,*N*-*M*+i}, *n*=*N*/*M* is the number of detector units in each block, S_1, S_2, .., S_M are the block. In each block, the forward and backward operation can be computed parallel. However, the blocks cannot be computed parallel, and need to be computed one by one. This makes the approximate parallel time-efficient.

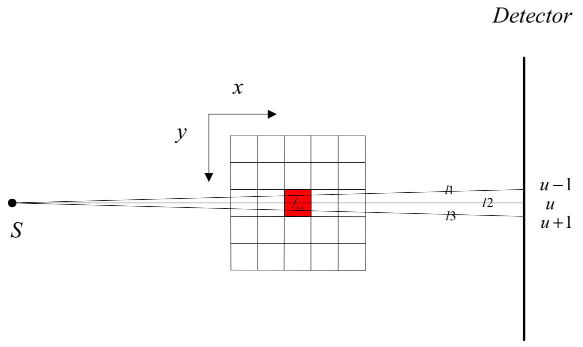

Fig. 1. The three rays can't be computed parallel.

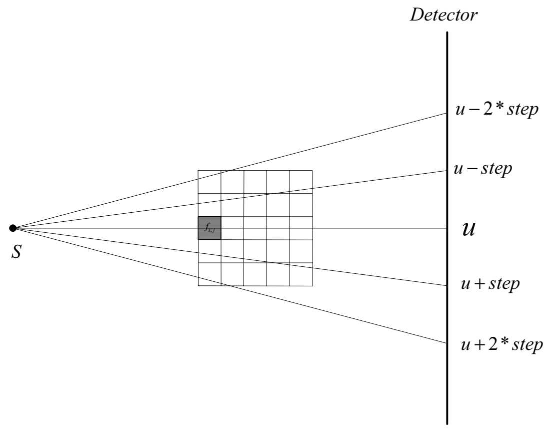

Fig. 2. The five rays can be computed paralle.

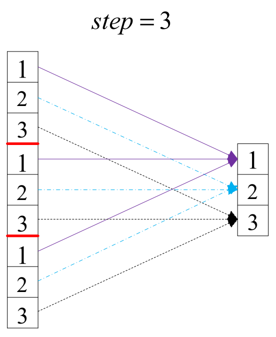

Fig. 3. Allot the 1D projection.

As described in Figure 3, if we assume $N_{block}=3$, the detector units are allotted into three blocks by the following rule: the detector number is ranged from 1 to N, units with number $3*n+1$ (n=0,1,...) are allotted into the first block, units with number $3*n+2$ (n=0,1,...) are allotted into the second block, and so on.

3.2. 3D cone beam CT case

According to Figure 4, the X-ray source moves in an ideal circle on the around z-axis, the (s, t) -system signifies the rotated (x, y)-system, θ is the angle of rotation. The (u, v)-system denotes the detector projection 2-D-coordinate system. Source position s, the origin o and the detector center-point D are in the same line. SDD is the constant source-to-detector distance. SAD is the constant source-to-z-axis distance.

It is easy to extend the 2D fan beam case to a 3D cone beam case. Similarly, *u_step* denotes the step in u direction, *v_step* means the step in v direction. We allot the detector units into *u_step*v_step* blocks, allot the detector units into patches with size of *u_step*v_step,* and for each patch, units are allotted to the corresponding location in the block.

We assume the projection as P_{uv} $(1\leq u \leq nu,\ 1\leq v \leq nv)$, nu is the dimension in u-direction and nv is the dimension in v-direction. Then, the blocks are $Block_{mn}$ $(1\leq m \leq u\_step,\ 1\leq n \leq v\_step)$, the detector pixel P_{ij} $(i=k*u\_step+m,\ j=l*v\_step+n)$ is allotted into the block $Block_{mn}$, where $k=0,1,2,\ldots,nu/u\_step-1$, $l=0,1,2,\ldots,nv/v\_step-1$.

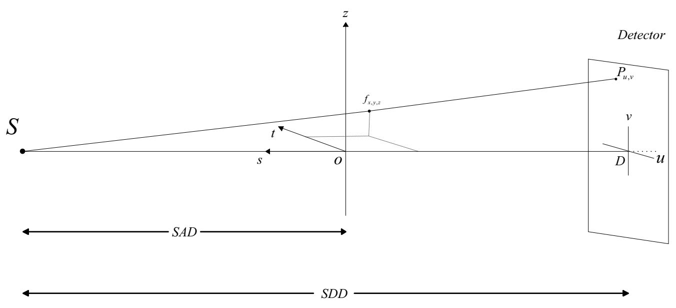

Fig. 4. Ideal and original geometry.

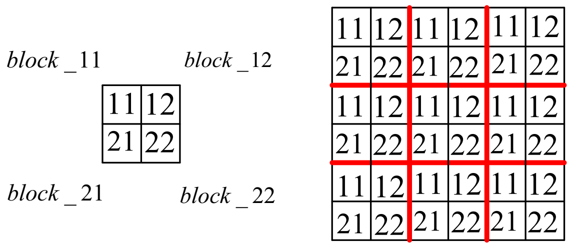

Fig. 5. Allot the 2D projection.

As shown in Figure 5, we assume *u_step=2*, *v_step=2*, so the block dimensions are *2*2*. We name the four blocks as *block_11*, *block_12*, *block_21*, *block_22*. The detector projection is split into patches with dimensions *2*2*. Then, the detector unit in each patch is allotted into corresponding blocks.

4. Experiments and discussion

4.1. Implementation flowchart

The parallel implementation for cone beam geometry over a circular scanning trajectory is discussed in this section.

For each angle θ:

(1) Calculate *u_step* and *v_step*;
(2) Allot the projections into *u_step*v_step* block;
(3) For each block, perform reconstruction on the GPU.

Algorithm 2: CUDA kernel

```
%ART_kernel----for each ray
    $error = p^i - \sum_{n=1}^{N} w_{in} f_n^{(k)}$
        for j = 1 : N do
            $f_j^{(k+1)} = f_j^{(k)} + l \frac{w_{ij}}{\sum_{n=1}^{N} w_{in}^2} error$
            Positivity constraint :
            $f_j^{(k+1)} = \begin{cases} f_j^{(k+1)} & f_j^{(k+1)} >= 0 \\ 0 & f_j^{(k+1)} < 0 \end{cases}$
        end for j
```

As implied in Eq. (3), in the CUDA parallel kernels, each thread is launched for each ray. This calculates the projection value of this ray, updating the voxels that the ray passes through.

4.2. Experiments on shepp-logan phantom

A NVIDIA K4000 card is used in this paper. The GPU card has a total of 768 processor cores, and is equipped with 3 GB GDDR5 memory.

We test our reconstruction algorithm with the shepp-logan phantom [13]. The phantom is generated with dimensions of 256*256*256 voxels with a voxel size of 0.5*0.5*0.5 mm^3. The x-ray imager is modeled as an array of 960*768 with detector size of 244*195 mm^2. The source-to-axes distance is 700 mm and the source-to-detector distance is 1000 mm. To estimate the computation efficiency, we compare the computation time between GPU framework implementation and CPU framework implementation. The forward-projection and backward-projection procedures are enforced by Joseph's ray tracing algorithm [14].

We use 60 projections to complete reconstruction. In all cases, the projections are taken along equally spaced angles, which cover an entire 360 degree rotation.

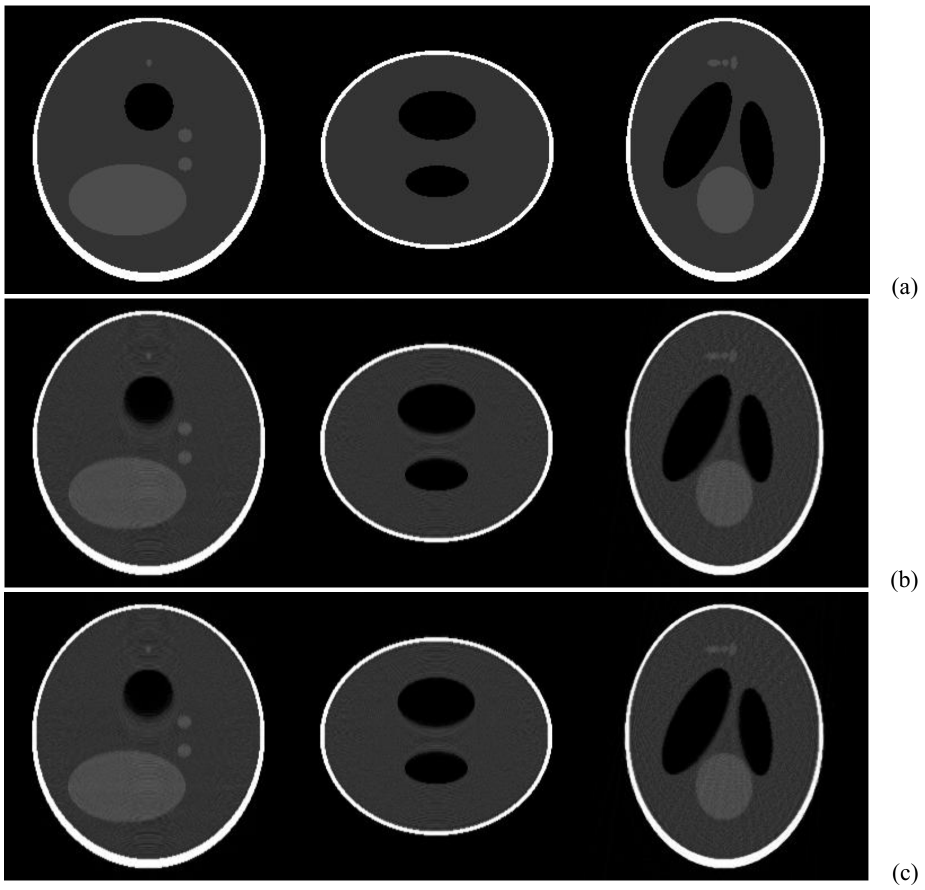

Fig. 6. Results of 3D shepp-logan phantom. Row (a) shows the standard digital image to be reconstructed; Row (b)-(c) shows the reconstructed image by CPU framework and GPU framework. The first through third columns show the coronal, sagittal, and transaxial views, respectively.

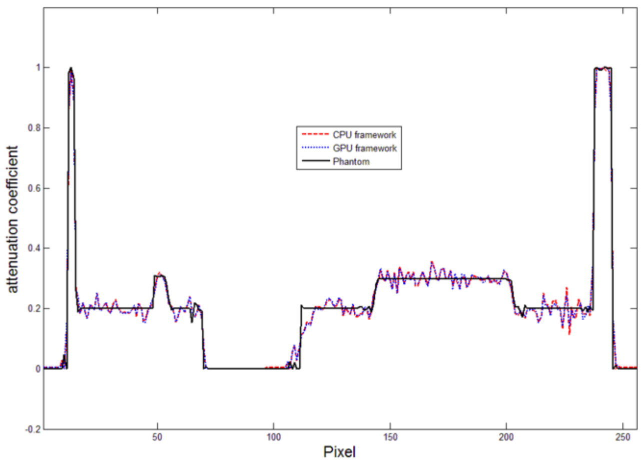

Fig. 7. Horizontal profiles through the center of coronal view images.

Table 1
The execution time needed for reconstruction

Computation Time t (sec)	CPU (per projection, average）	GPU (per projection, average）
	40s	2s

Slices of reconstructed images after 5 iterations are shown in Figure 6. Together with the corresponding slices of the phantom, columns 1–3 show the coronal, sagittal, and transaxial views of the images, respectively. Row (a) in Figure 6 shows the digital phantom images. Row (b) shows the 3D reconstructed image by the CPU framework. Row (c) shows the 3D reconstructed image by the GPU framework. Because the model is very simple, it does not support high-quality images. The outcome of the GPU framework is nearly the same with the outcome of the CPU framework. Horizontal profiles through the center of the coronal view images are plotted in Figure 7, which further illustrates the similarity of the different reconstruction frameworks.

The ART implementation in CUDA 5.0 wastes approximately 2.0 second per angle per update to complete the forward-projection and the back-projection.

As we can see from Table 1, compared with the computational time of almost 40 s in CPU implementation, our GPU implementation on a NVIDIA Tesla C1060 GPU card has tremendously enhanced efficiency (~20 times faster).

From the experiment above, we can see that the GPU framework achieves the same outcome, but in much less time than the CPU framework.

5. Conclusion

In this paper, we have developed a block-wise approximate parallel implementation method for the ART algorithm on a CUDA-enable GPU.

The result on a shepp-logan phantom indicates that our algorithm has improved the time efficiency by a factor of 20 over the existing CPU implementation method. These findings make the iterative CBCT reconstruction approach applicable in clinical environments.

Due to its reconstruction time of less than 10 minutes, our GPU implementation is already applicable for specific usage in the clinical environment.

Acknowledgments

This work is supported in part by grants from National Natural Science Foundation of China (NSFC: 81171402), NSFC Joint Research Fund for Overseas Research Chinese, Hong Kong and Macao Young Scholars (30928030), National Basic Research Program 973 (2010CB732606) from Ministry of Science and Technology of China, Guangdong Innovative Research Team Program (No. 2011S013) of China, Science and Technological Program for Dongguan's Higher Education, Science and Research, and Health Care Institutions (Grant No. 2011108101001), and Comprehensive Strategic Cooperation Project of Guangdong province and Chinese academy of sciences (Grant No. 2011B090300079).

References

[1] L.A. Feldkamp, L.C. Davis and J.W. Kress, Practical cone-beam algorithm, Journal of the Optical Society of America A **1** (1984), 612-619.

[2] R. Gordon, R. Bender and G.T. Herman, Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and x-ray photography, Journal of Theoretical Biology **29** (1970), 471-482.

[3] A.H. Andersen and A.C. Kak, Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm, Ultrason Imaging **6** (1984), 81-94.

[4] P. Gilbert, Iterative methods for the three-dimensional reconstruction of an object from projections, Journal of Theoretical Biology **36** (1972), 105-117.

[5] C. Badea and R. Gordon, Experiments with the nonlinear and chaotic behaviour of the multiplicative algebraic reconstruction technique (MART) algorithm for computed tomography, Physics in Medicine and Biology **49** (2004), 1455-1474.

[6] K. Lange and R. Carson, EM reconstruction algorithms for emission and transmission tomography, Journal of Computer Assisted Tomography **8** (1984), 306-316.

[7] S.H. Manglos, G.M. Gagne, A. Krol, F.D. Thomas and R. Narayanaswamy, Transmission maximum-likelihood reconstruction with ordered subsets for cone beam CT, Physics in Medicine and Biology **40** (1995), 1225-1241.

[8] K. Mueller, R. Yagel and J.J. Wheller, A fast implementations of algebraic methods for three-dimensional reconstruction from cone-beam data, IEEE Transactions on Medical Imaging **18** (1999), 538-548.

[9] H. Guan and R. Gordon, Computed tomography using ART with different projection access schemes, Physics in Medicine and Biology **41** (1996), 1727-1743.

[10] K. Mueller, R. Yagel and J.J. Wheller, Fast implementations of algebraic methods for the 3D reconstruction from conebeam data, IEEE Transaction Medical Imaging **18** (1999), 538-548.

[11] Guillem Pratx and Lei Xing, GPU computing in medical physics: A review, Medical Physics **38** (2011), 2685-2698.

[12] S. Kaczmarz, Angenaherte anosung von systemen linear gleichungen, Bulletin International de l'Académie Polonaise des Sciences et des Lettres A **35** (1937), 355-357.

[13] L.A. Shepp and B.F. Logan, The Fourier reconstruction of a head section, IEEE Transactions on Nuclear Science **21** (1974), 21-43.

[14] P.M. Joseph, An improved algorithm for reprojecting rays through pixel images, IEEE Transaction Medical Imaging **1** (1982), 192-198.