Accelerating Separable Footprint (SF) Forward and Back Projection on GPU

Xiaobin Xie[a,b], Madison G. McGaffin[c], Yong Long[d], Jeffrey A. Fessler[c], Minhua Wen[b], and James Lin[b]

[a]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China

[b]Center for High Performance Computing, Shanghai Jiao Tong University, Shanghai, 200240, China

[c]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA

[d]University of Michigan - Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, 200240, China

ABSTRACT

Statistical image reconstruction (SIR) methods for X-ray CT can improve image quality and reduce radiation dosages over conventional reconstruction methods, such as filtered back projection (FBP). However, SIR methods require much longer computation time. The separable footprint (SF) forward and back projection technique simplifies the calculation of intersecting volumes of image voxels and finite-size beams in a way that is both accurate and efficient for parallel implementation. We propose a new method to accelerate the SF forward and back projection on GPU with NVIDIA's CUDA environment. For the forward projection, we parallelize over all detector cells. For the back projection, we parallelize over all 3D image voxels. The simulation results show that the proposed method is faster than the acceleration method of the SF projectors proposed by Wu and Fessler.[13] We further accelerate the proposed method using multiple GPUs. The results show that the computation time is reduced approximately proportional to the number of GPUs.

Keywords: Statistical image reconstruction (SIR), X-ray CT, forward and back projection, separable footprint (SF), GPU, CUDA

1. INTRODUCTION

Statistical image reconstruction (SIR) methods for X-ray CT improve the ability to produce high-quality and accurate images, while greatly reducing radiation dosages. However, SIR methods operate with one major drawback. They need long computation time to process the scanned data and reconstruct a diagnostically useful image. SIR methods iteratively find the image that best fits the measurement, according to the system physical model, the measurement statistical model and prior information about the object.

Most SIR methods require one forward projection and one back projection in each iteration. These operations are the primary computational bottleneck in SIR methods, especially in 3D image reconstruction. Accelerating forward and back projection is crucial to fast implementation of SIR methods. Numerous approaches have been proposed to accelerate the forward and back projection using Graphics Processing Unit (GPU). Gao proposed fast and highly parallelizable algorithms for X-ray transform and its adjoint for the infinitely narrow beam in both 2D and 3D, but only extended these fast algorithms to the finite-size beam in 2D.[1] To extend these fast algorithms to the finite-size beam in 3D, an efficient 3D formula to compute the intersection volume of the finite-size beam with each nontrivially intersecting voxel needs to be supplied.[1] Xie et al.[2] proposed a Fixed Sampling Number Projection (FSNP) strategy to ensure the operation synchronization in ray-driven forward

Further author information: (Send correspondence to Yong Long)
Yong Long: E-mail: yong.long@sjtu.edu.cn

Medical Imaging 2017: Physics of Medical Imaging, edited by Thomas G. Flohr, Joseph Y. Lo, Taly Gilat Schmidt, Proc. of SPIE Vol. 10132, 101322S · © 2017 SPIE
CCC code: 1605-7422/17/$18 · doi: 10.1117/12.2252010

projection[3] implementation on GPU, and parallelized the voxel-driven back projection method.[4] Bippus et al.[5] accelerated the blob-driven back projection proposed by Ziegler et al.[6] using GPU, but implemented a ray-driven forward projection to avoid synchronization effort for writing operations due to overlapping blob footprints on the detector. This method implemented asymmetric projector/backprojector pair, which results in an increasing mismatch in the sinogram and image space as iteration continues.[6] Liu et al.[7] implemented a branchless Distance-Driven (DD) algorithm[8] that is highly parallelizable and amenable to vectorization on GPUs.

Separable footprint (SF) forward and back projection technique[9] is more accurate[14] than the popular distance-driven (DD) method,[10] and has comparable computation speed as the DD method. Many researchers have implemented the SF projector on GPU to accelerate their own projects.[11,12] Wu and Fessler[13] implemented the SF forward and back projector on GPU with NVIDIA's CUDA environment. For the SF forward projection, to avoid writing conflict due to parallelizing arbitrary image voxels, they grouped the image voxels according to the smallest transaxial coordinates of their footprints and parallelized over voxel groups whose footprints do not overlap. For the SF back projection, they parallelized over all 3D image voxels to prevent read-write problems because each thread works on disjoint image voxels. Papenhausen et al. proposed a method that is able to tune high level implementation details by using the ant colony optimization algorithm to find the optimal implementation in a relatively short amount of time.[14] Using the presented framework, they optimized the performance of the GPU accelerated SF backprojection implementation by Wu and Fessler.[13]

We propose a GPU acceleration method to SF forward and back projection using CUDA. For the SF forward projection, we parallelize over detector cells. From each detector cell, we trace image voxels which contribute to this detector cell. For the SF back projection, we parallelize over all 3D image voxels. We demonstrate that our new method has better performance in terms of speed than the acceleration method proposed by Wu and Fessler.

This paper is organized as follows. Section 2.1 describes the SF projector. Section 2.2 and 2.3 present GPU acceleration methods for the SF forward and back projection respectively. Section 3 gives results and Section 4 describes conclusions.

2. METHODS

2.1 SF Projector

Any 3D method for forward and back projection in X-ray CT can be described as:[13]

$$g(s,t,\beta) = \sum_{x,y,z} a(s,t,\beta;x,y,z) f(x,y,z), \tag{1}$$

$$b(x,y,z) = \sum_{s,t,\beta} a(s,t,\beta;x,y,z) g(s,t,\beta), \tag{2}$$

where $f(x,y,z)$ and $b(x,y,z)$ denote the pixel values and back projection values for 3D image locations (x,y,z) respectively, $g(s,t,\beta)$ denotes the projection values in the detector plane, s and t denote the transaxial and axial directions of the 2D detector plane respectively, β denotes the angle of the source point counter-clockwise from the y axis, and $a(s,t,\beta;x,y,z)$ is the blurred footprint function.

The SF method approximates the 2D blurred footprint function $a(s,t,\beta;x,y,z)$ as the separable product of 1D blurred footprint functions in the axial and transaxial directions,[9] i.e.,

$$a(s,t,\beta;x,y,z) = v(s,t,\beta)u(\beta;x,y)F_1(s,\beta;x,y)F_2(t,\beta;x,y,z), \tag{3}$$

where $F_1(s,\beta;x,y)$ is the blurred footprint in the transaxial direction, $F_2(t,\beta;x,y,z)$ is the blurred footprint in the axial direction, and $v(s,t,\beta)$ and $u(\beta,x,y)$ are the amplitude functions as described in[9] as the "A2" method. We focus on GPU acceleration of the SF-TR projector where $F_1(s,\beta;x,y)$ is the integration of a trapezoid footprint function and a rectangular detector blur function and $F_2(t,\beta;x,y,z)$ is the integration of a rectangular footprint function and a rectangular detector blur function. The acceleration method described in this paper can be easily extended to the SF-TT projector[9] where $F_2(t,\beta;x,y,z)$ is the integration of a trapezoid footprint function and a rectangular detector blur function.

2.2 SF Forward Projection

Combining (1) and (3), the SF forward projection is

$$g(s,t,\beta) = v(s,t,\beta)\sum_{x,y} F_1(s,\beta;x,y)[\sum_{z} F_2(t,\beta;x,y,z)f(x,y,z)]u(\beta;x,y). \tag{4}$$

For each projection view β, we compute the SF forward projection in three CUDA kernels: *transaxial_footprints*, *axial_sum*, and *ray_sum*.

In the *transaxial_footprints* kernel, we use $N_x \times N_y$ threads to compute the blurred footprint $F_1(s,\beta;x,y)$, and the smallest and largest vertices of the trapezoid footprint function in the s direction: $s_{min}(\beta;x,y)$ and $s_{max}(\beta;x,y)$. N_x and N_y are the number of image voxels in the x and y directions respectively. For a typical CT geometry, an image voxel contributes to at most ten detector cells in the s direction, so we use an array of $10 \times N_x \times N_y$ to store the blurred footprint $F_1(s,\beta;x,y)$ values.

In the *axial_sum* kernel, we use $N_t \times N_x \times N_y$ threads to compute the projection values in the axial direction for a single projection view. N_t is the number of detector cells in the axial direction. Each thread finds image voxels that contribute in the axial direction to the associated $(t;x,y)$ location, computes the blurred footprints $F_2(t,\beta;x,y,z)$ of these image voxels, computes projection values in the axial direction using (5), and weights the projection values using (6), i.e., the inner-most sum in (4):

$$p_1(t,\beta;x,y) = \sum_{z\in G_z} F_2(t,\beta;x,y,z)f(x,y,z), \quad G_z = \{z : t \in [t_{min}(\beta;x,y,z), t_{max}(\beta;x,y,z)]\}, \tag{5}$$

$$p_2(t,\beta;x,y) = p_1(t,\beta;x,y)u(\beta;x,y). \tag{6}$$

Each thread finds the contributing image voxels according to their footprints and the t location associated with this thread, i.e., $t \in [t_{min}(\beta;x,y,z), t_{max}(\beta;x,y,z)]$. The projection values $p_2(t,\beta;x,y)$ in the axial direction are stored in an array of size $N_t \times N_x \times N_y$.

In the *ray_sum* kernel, we use $N_t \times N_s$ threads to compute the forward projection values for all detector cells where N_s is the number of detector cells in the transaxial direction. Each thread is assigned to one detector cell. Each thread finds image voxels that contribute to the associated detector cell, accumulates contributions of these voxels using (7), weights the projection value using (8), and writes the projection value to the corresponding location in a $N_t \times N_s$ array, i.e.,

$$g^{'}(s,t,\beta) = \sum_{(x,y)\in G} F_1(s,\beta;x,y)p_2(t,\beta;x,y), \quad G = \{(x,y) : s \in [s_{min}(\beta;x,y), s_{max}(\beta;x,y)]\}, \tag{7}$$

$$g(s,t,\beta) = v(s,t,\beta)g^{'}(s,t,\beta). \tag{8}$$

Each thread needs to find (x,y) locations whose footprints cover the associated detector cell in the transaxial direction. Depending on projection angle β which determines the relative location between the X-ray source and x (or y) axis, we determine a primary direction and a secondary direction. If the source is closer to the y axis, the primary direction is y and the secondary direction is x. If the source is closer to the x axis, the primary direction is x and the secondary direction is y. Each thread advances along the primary direction from its associated detector cell towards the source. Along each row of the primary direction, each thread looks the intersecting voxel and its neighbours along the secondary direction, and checks if the (x,y) locations cast footprints that cover this thread's detector cell. If $s \in [s_{min}(\beta;x,y), s_{max}(\beta;x,y)]$, then the (x,y) location is accumulated using (7).

We optimize the memory usage in the *ray_sum* kernel. The $s_{min}(\beta;x,y)$ and $s_{max}(\beta;x,y)$ values of all the (x,y) locations are computed in the *transaxial_footprints* kernel and stored in global memory. In the *ray_sum* kernel, we load the $s_{min}(\beta;x,y)$ and $s_{max}(\beta;x,y)$ values from global memory into shared memory, and then read these values from shared memory when finding contributing (x,y) locations. This memory usage optimization speeds up the *ray_sum* kernel which is the most time-consuming kernel.

Algorithm 1 summaries the pseudo-code for GPU implementation of the SF forward projection. The CUDA kernels execute in parallel.

Algorithm 1 PSEUDO-CODE FOR GPU IMPLEMENTATION OF THE SF-TR FORWARD PROJECTOR WITH THE A2 METHOD (SF-TR-A2)

CPU loop: for each projection angle β:

(a) **CUDA kernel transaxial_footprints**: parfor each x and y ($N_x \times N_y$):

i. Compute and store $F_1(s, \beta; x, y)$.

ii. Compute and store $s_{min}(\beta; x, y)$ and $s_{max}(\beta; x, y)$.

(b) **CUDA kernel axial_sum**: parfor each t, x and y ($N_t \times N_x \times N_y$):

i. Compute $p_1(t, \beta; x, y)$ in (5)

ii. Compute and store $p_2(t, \beta; x, y)$ in (6)

(c) **CUDA kernel ray_sum**: parfor each t and s ($N_t \times N_s$):

i. Compute $g^{'}(s, t, \beta)$ in (7):

a. Trace along the primary direction (x or y). For each row of the primary direction (x or y)

- look intersecting voxels and its neighbours along the secondary direction (x or y).
- If $s_{min}(\beta; x, y) \leq s \leq s_{max}(\beta; x, y)$ then accumulate this (x, y) location using (7).

ii. Compute $g(s, t, \beta)$ in (8).

2.3 SF Back Projection

Combining (2) and (3), the SF back projection is

$$b(x, y, z) = \sum_{\beta} u(\beta; x, y) \sum_{t} F_2(t, \beta; x, y, z) [\sum_{s} F_1(s, \beta; x, y) g(s, t, \beta) v(s, t, \beta)]. \tag{9}$$

For each projection view β, we compute the SF back projection in three CUDA kernels: *transaxial_footprints*, *transaxial_sum* and *voxel_sum*. The *transaxial_footprints* kernel is the same as that of the forward projection.

In the *transaxial_sum* kernel, we use $N_t \times N_x \times N_y$ threads to compute the back projection values in the transaxial direction. Each thread finds detector cells that contribute in the transaxial direction to the associated $(t; x, y)$ location, accumulates contributions of these detector cells using (10), weights the back projection value using (11), and writes the back projection value to the corresponding element in a $N_t \times N_x \times N_y$ array, i.e., the inner-most sum in (9):

$$b_1(t, \beta; x, y) = \sum_{s} F_1(s, \beta; x, y) g(s, t, \beta) v(s, t, \beta), \quad \forall s \in [s_{min}(\beta; x, y), s_{max}(\beta; x, y)], \tag{10}$$

$$b_2(t, \beta; x, y) = b_1(t, \beta; x, y) u(\beta; x, y). \tag{11}$$

Each thread finds the contributing detector cells according to their s coordinates and footprint of the (x, y) location associated with this thread, i.e., $s \in [s_{min}(\beta; x, y), s_{max}(\beta; x, y)]$. The *transaxial_sum* kernel is the adjoint of the *ray_sum* kernel.

In the *voxel_sum* kernel, we use $N_x \times N_y \times N_z$ threads to compute the back projection values for image voxels. Each thread finds detector cells that contribute in the axial direction to the associated (x, y, z) location, calculates a 3D image by back projecting each projection view using (12), and then accumulates those back-projected images using (13), i.e.,

$$b(\beta; x, y, z) = \sum_{t} b_2(t, \beta; x, y) F_2(t, \beta; x, y, z), \quad \forall t \in [t_{min}(\beta; x, y, z), t_{max}(\beta; x, y, z)], \tag{12}$$

$$b(x, y, z) \mathrel{+}= b(\beta; x, y, z). \tag{13}$$

Each thread finds the contributing detector cells according to their t coordinates and vertices of footprint functions of the image voxel associated with this thread, i.e., $t \in [t_{min}(\beta; x, y, z), t_{max}(\beta; x, y, z)]$. The *voxel_sum* kernel is the adjoint of the *axial_sum* kernel.

Algorithm 2 summaries the pseudo-code for GPU implementation of the SF back projection. The CUDA kernels execute in parallel.

Algorithm 2 PSEUDO-CODE FOR GPU IMPLEMENTATION OF THE SF-TR BACK PROJECTOR WITH THE A2 METHOD (SF-TR-A2)

1. Initialize image data array to zero: $b(x, y, z) = 0$
2. CPU loop: for each projection angle β:
   (a) **CUDA kernel transaxial_footprints**: parfor each x and y ($N_x \times N_y$):
      i. Compute and store $F_1(s, \beta; x, y)$.
      ii. Compute and store $s_{min}(\beta; x, y)$ and $s_{max}(\beta; x, y)$.
   (b) **CUDA kernel transaxial_sum**: parfor each t, x and y ($N_t \times N_x \times N_y$):
      i. Compute $b_1(t, \beta; x, y)$ in (10).
      ii. Compute and store $b_2(t, \beta; x, y)$ in (11).
   (c) **CUDA kernel voxel_sum**: parfor each x, y and z ($N_x \times N_y \times N_z$):
      i. Compute $b(\beta; x, y, z)$ in (12).
      ii. Compute $b(x, y, z)$ in (13).

3. RESULTS

To evaluate our proposed GPU acceleration method of the SF-TR forward and back projectors, we compared it with the GPU method proposed by Wu and Fessler[13] and the original CPU implementation.[9] We call the GPU acceleration method proposed by Wu and Fessler[13] as the WF method hereafter. We tested the two GPU methods that are both implemented in CUDA and the original CPU implementation in an ANSI C routine on a server with two 14-core 2.30GHz Intel Xeon E5-2695 v3 CPUs and two NVIDIA Tesla K80 graphic cards (one K80 graphic card has two GPUs). We simulated the geometry of a GE LightSpeed X-ray CT system with an arc detector of $N_s = 888$ detector columns for $N_t = 64$ detector rows with $N_\beta = 984$ views over 360° for a 3D object of size N_x= 512, $N_y = 512$ and $N_z = 64$ with a resolution of $\Delta_X = \Delta_Y = 0.9766$ and $\Delta_Z = 0.625$. The size of each detector cell was $\Delta_S \times \Delta_T = 1.0239 \times 1.0964\text{mm}^2$. The source to detector distance was 949.075mm, and the source to rotation center distance was 541mm.

Table 1 summaries the speed and accuracy comparison of the proposed and the WF method for axial CT. We measured the accuracy using the normalized root mean square (NRMS) error:$\sqrt{\frac{1}{N}\sum_{j=1}^{N}(\frac{\hat{y}_j - y_j}{y_j})^2}$ where y_j denotes the projection value by the original CPU implementation[9] and $\hat{y}_j$ denotes the projection value by the proposed or the WF method. The proposed method is about 1.7 times faster than the WF method for both the forward and back projection. For the back projection, the proposed method has smaller NRMS errors than the WF method. The WF method used only $3 \times N_x \times N_y \times N_t$ threads rather than $N_x \times N_y \times N_z$ to implement (12). For the forward projection, the WF method runs a CPU loop for ten times and uses $N_t \times [\frac{N_s}{10}]$ threads to compute projection values of the associated detector cells in each loop, while the proposed method uses $N_t \times N_s$ threads to compute projection values for all detector cells.

	method	Calculation Time (s)	NRMS error(%)
forward projection	proposed method	7.2	5.4×10^{-7}
	WF method	12	5.5×10^{-7}
back projection	proposed method	4.6	2.9×10^{-4}
	WF method	7.7	1.4×10^{-2}

Table 1. Speed and accuracy comparison of the proposed and the WF acceleration method of the SF-TR projectors for axial CT

Table 2 shows the computation time of each CUDA kernel in the proposed acceleration method for the SF-TR forward and back projection. Since the *axial_sum* and *ray_sum* kernel for the forward projection are the adjoints of the *voxel_sum* and the *transaxial_sum* kernel for the back projection respectively, we group them together in Table 2. The *axial_sum* and *voxel_sum* kernel have similar computation time, but the *ray_sum* kernel requires

longer computation time than the *transaxial_sum* kernel because the *transaxial_sum* kernel uses more threads than the *ray_sum* kernel and the *ray_sum* kernel needs to trace contributing voxels in x and y directions.

Calculation Time (s)	Forward Projection	Back Projection
transaxial_footprints	0.17	0.18
axial_sum(voxel_sum)	2.0	1.8
ray_sum(transaxial_sum)	5.0	2.5

Table 2. Computation time of each CUDA kernel in the proposed method

Table 3 shows the computation time and accuracy of the proposed acceleration method and the WF method for helical CT with 8 turns. The test object has a size of $N_x = 512$, $N_y = 512$ and $N_z = 512$. All turns share the same $F_1(s, \beta; x, y)$, $s_{min}(\beta; x, y)$ and $s_{max}(\beta; x, y)$ for a projection angle β of different turns. We only need to execute the *transaxial_footprints* kernel once for a β and use its results for the same β of other turns. The results show that the proposed method is 1.3 times faster for the forward projection and 1.6 times faster for the back projection than the WF method.

	method	Calculation Time (s)	NRMS error(%)
forward projection	proposed method	57	1.7×10^{-6}
	WF method	73	1.9×10^{-6}
back projection	proposed method	38	3.0×10^{-4}
	WF method	61	1.5×10^{-2}

Table 3. Speed and accuracy comparison of the proposed and the WF acceleration method of the SF-TR projectors for helical CT with 8 turns

We used multiple GPUs to further accelerate the proposed method. For the forward projection, we assigned each GPU a different subset of the views. For the back projection, we divided the object along the x or y direction into different subsets and assigned each GPU a subset. Table 4 and Table 5 summarize the computation time of the proposed method using multiple GPUs for axial and helical CT with 8 turns respectively. Due to the large bandwidth of NVIDIA Tesla K80 GPUs, the time for the transfer of data between CPUs and GPUs is very short. The computation time is reduced approximately proportional to the number of GPUs.

Calculation Times(s)	Forward Projection	Back Projection
one GPU	7.2	4.6
two GPUs	3.8	2.4
three GPUs	2.5	1.6
four GPUs	1.9	1.2

Table 4. Computation time of the proposed method using multiple GPUs for axial CT

Calculation Times(s)	Forward Projection	Back Projection
one GPU	57	38
two GPUs	29	19
three GPUs	19	13
four GPUs	15	10

Table 5. Computation time of the proposed method using multiple GPUs for helical CT with 8 turns

Fig. 1 shows the wall time of original CPU implementation[9] of the SF-TR forward and back projection for the axial CT simulation. Because of the "hyperthreading" of Intel Xeon E5-2695 v3 CPUs, we used up to 56 POSIX threads. Fig. 2 shows the speedup of the original CPU implementation with 1 to 56 threads. The speedup saturates at 28 threads which is the number of CPU cores. The shortest time is 4.9 seconds using 53 threads for the forward projection and 5.1 seconds using 54 threads for the back projection. Using two GPUs, the proposed method takes 3.8 seconds for the forward projection and 2.4 seconds for the back projection, which is 1.3 times faster than the original CPU implementation for the forward projection and 2.1 times faster for the back projection respectively. One NVIDIA Tesla K80 graphic card with 2 GPUs costs about 5000 dollars and two 14-core 2.30GHz Intel Xeon E5-2695 v3 CPUs cost about 4900 dollars. With similar cost, the proposed method is able to achieve much faster computational speed.

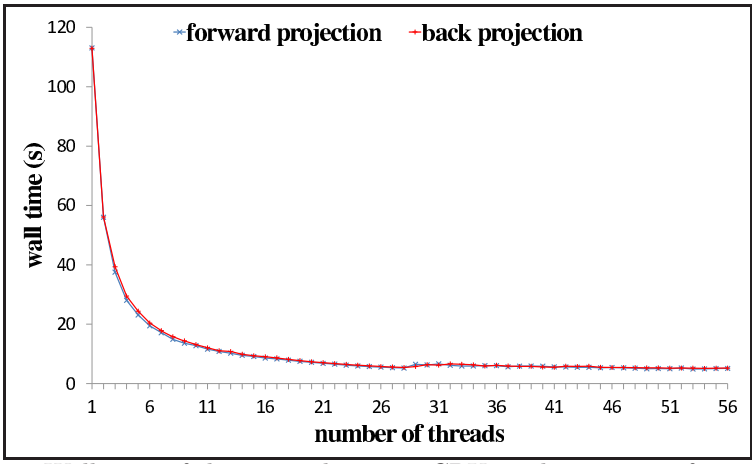

Figure 1. Wall time of the original 28-core CPU implementation for axial CT.

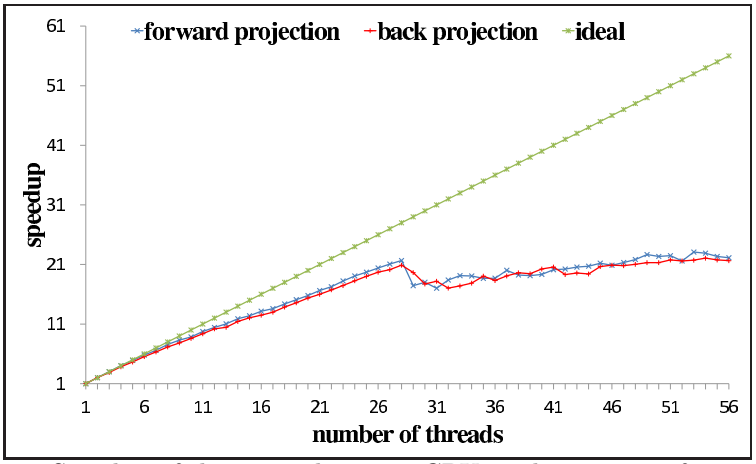

Figure 2. Speedup of the original 28-core CPU implementation for axial CT.

4. CONCLUSIONS

We proposed an acceleration method of the SF forward and back projection[9] on GPU with NVIDIA's CUDA environment. We parallelized over all detector cells for the forward projection and parallelized over all 3D image voxels for the back projection. We used shared memory to optimize the memory usage in the forward projection, which speeds up the calculation. Compared with GPU acceleration method proposed by Wu and Fessler,[13] our proposed method increased the degree of parallelism for the forward projection. Simulation results have shown that the proposed method is faster than the acceleration method proposed by Wu and Fessler. We used multiple GPUs to further accelerate the proposed method and tested it on 4 NVIDIA Tesla K80 GPUs. The computation time is reduced approximately proportional to the number of GPUs. We will investigate methods to further accelerate the SF forward projection in the future.

ACKNOWLEDGMENTS

The authors would like to thank the Center for High Performance Computing of Shanghai Jiao Tong University for providing the test server with CPUs and GPUs. This work was partially supported by SJTU-UM Collaborative Research Program, Shanghai Pujiang Talent Program (15PJ1403900), NSFC (61501292) and Returned Overseas Chinese Scholars Program.

REFERENCES

[1] Gao, H., "Fast parallel algorithms for the x-ray transform and its adjoint," *Medical physics* **39**(11), 7110-7120 (2012).

[2] Xie, L., Hu, Y., Yan, B., Wang, L., Yang, B., Liu, W., Zhang, L., Luo, L., Shu, H., and Chen, Y., "An Effective CUDA Parallelization of Projection in Iterative Tomography Reconstruction," *PloS one* **10**(11), e0142184 (2015).

[3] Siddon, R. L., "Fast calculation of the exact radiological path for a three-dimensional CT array," *Medical physics* **12**(2), 252-255 (1985).

[4] Peters, T. M., "Algorithms for fast back- and re-projection in computed tomography," *IEEE transactions on nuclear science* **28**(4), 3641-3647 (1981).

[5] Bippus, R. D., Koehler, T., Bergner, F., Brendel, B., Hansis, E., and Proksa, R., "Projector and backprojector for iterative CT reconstruction with blobs using CUDA," *Fully 3D 2011: 11th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Potsdam, Germany*, (2011).

[6] Ziegler, A., Kohler, T., Nielsen, T., and Proksa, R., "Efficient projection and backprojection scheme for spherically symmetric basis functions in divergent beam geometry," *Medical physics* **33**(12), 4653-4663 (2006).

[7] Liu, R., Fu L., De Man B., and Yu H., "GPU Acceleration of Branchless Distance Driven Projection and Backprojection," *Proc. 4th Intl. Mtg. on image formation in X-ray CT*, 229-232 (2016).

[8] Basu, S., and De Man, B., "Branchless distance driven projection and backprojection," *Electronic Imaging 2006,* 60650Y-60650Y (2006).

[9] Long, Y., Fessler, J. A., and Balter, J. M., "3D forward and back-projection for X-ray CT using separable footprints," *IEEE Trans. Med. Imag.* **29**, 1839-50 (Nov. 2010).

[10] De Man, B., and Basu, S., "Distance-driven projection and backprojection in three dimensions," *Physics in medicine and biology* **49**(11), 2463 (2004).

[11] Gang, G. J., Stayman, J. W., Zbijewski, W., and Siewerdsen, J. H., "Task-based detectability in CT image reconstruction by filtered backprojection and penalized likelihood estimation," *Medical physics* **41**(8), 081902 (2014).

[12] Wang, A. S., Stayman, J. W., Otake, Y., Kleinszig, G., Vogt, S., Gallia, G. L., Khanna, A. J. and Siewerdsen, J. H., "Soft-tissue imaging with C-arm cone-beam CT using statistical reconstruction," *Physics in medicine and biology* **59**(4), 1005 (2014).

[13] Wu, M., and Fessler, J. A., "GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction," *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med.* **6**, 021911 (2011).

[14] Papenhausen, E., Zheng, Z., and Mueller, K., "Creating optimal code for GPU-accelerated CT reconstruction using ant colony optimization," *Medical physics* **40**(3), 031110 (2013).