Fast Cone-beam CT reconstruction using GPU

Giovanni Di Domenico[1]

[1]Dipartimento di Fisica e Scienza della Terra, via Saragat 1, 44122 Ferrara, Italy
[2]INFN - Sezione di Ferrara, via Saragat 1, 44122 Ferrara, Italy

DOI: http://dx.doi.org/10.3204/DESY-PROC-2014-05/35

Fast 3D cone-beam CT reconstruction is required by many application fields like medical CT imaging or industrial non-destructive testing. We have used GPU hardware and CUDA platform to speed-up the Feldkamp Davis Kress (FDK) algorithm, which permits the reconstruction of cone-beam CT data. In this work, we present our implementation of the most time-consuming step of FDK algorithm: back-projection. We also explain the required transformations to parallelize the algorithm for the CUDA architecture. Our FDK algorithm implementation in addition allows to do a rapid reconstruction, which means that the reconstruction is completed just after the end of data acquisition.

1 Introduction

There are a lot of application areas that benefit of the GPU computing, among them we find the image reconstruction in medical physics applications [1]. In the last years, the computational complexity of image reconstruction has increased with the progress in imaging systems and reconstruction algorithms. Moreover, fast image reconstruction is required in an imaging diagnostic department to allow the technologist to review the images while the patient is waiting or to obtain the output in real-time imaging applications. Many researchers have worked to accelerate Cone-Beam CT (CBCT) reconstruction using various accelerators, such as GPU [3, 4, 5, 6], Cell Broadband Engine (CBE) [7], and field programmable gate array (FPGA) [8]. The Common Unified Device Architecture (CUDA) [2] is a software development platform, invented by nVIDIA, that allows us to write and execute general-purpose application on graphics processing unit (GPU). Scherl et al. [4] have developed one of the first GPU-accelereted CT reconstruction using CUDA and they have compared the performance results of CUDA implementation with the CBE ones concluding that the CUDA-enable GPU is well suited for high-speed CBCT recontruction. In this paper, we propose a CUDA based method capable of accelerating CBCT reconstruction based on FDK algorithm, showing some optimization techniques for the backprojection step.

2 Theory

The ideal geometry for cone-beam CT, see Fig.1, is made by an x-ray source that moves on a circle through the x-y plane around z, the axis of rotation. The (u,v,w) denotes the rotated reference frame, where β is the angle of rotation, R_s is the radius of the source circle, while R_d is the source to detector distance. The line from the x-ray source through the origin O hits the

detector at the origin of its 2D-coordinate system (u,v) and is orthogonal to the detector plane. The CT measurement can be converted to a form that is closely modeled by a line integral

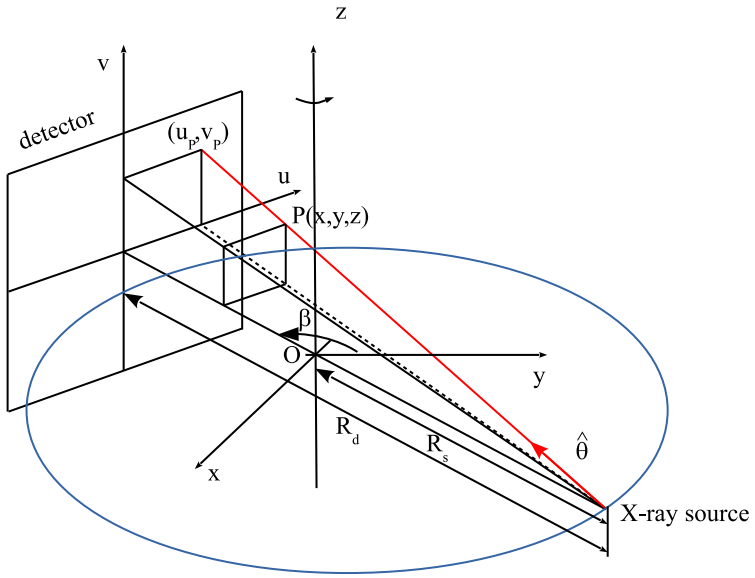

Figure 1: Cone-beam CT geometry.

through the continuous object function

$$g(u,v,\beta) = \int_0^\infty f(\vec{r}_o(\beta) + \alpha\hat{\theta})d\alpha \tag{1}$$

where $f(\vec{r})$ represents the object function, the x-ray attenuation coefficient, and the data function $g(u,v,\beta)$ is the line integral through the object in the direction of unit vector $\hat{\theta}$ from the source location $\vec{r}_o(\beta)$. For the model considered here, the detector is taken to be a flat panel array with bin location (u_p, v_p)

$$\vec{d}_o(u_p,v_p,\beta) = (R_d - R_s)(-sin\beta, cos\beta, 0) + u_p(cos\beta, sin\beta, 0) + v_p(0,0,1) \tag{2}$$

The goal of image reconstruction is to find $f(\vec{r})$ from the knowledge of $g(u,v,\beta)$.

2.1 FDK algorithm

The FDK algorithm [9] reconstructs the function $f(\vec{r})$ accurately only in the plane of trajectory. Outside of this plane, the algorithm approximate $f(\vec{r})$. The FDK algorithm applies a filtered backprojection technique to solve the reconstruction task in a computationally efficient way. Due to its efficiency it has been implemented successfully on almost ever commercially available medical CT image system and still maintain its state of the art status in modern computed tomography. The FDK algorithm is organized in three steps:

1. cosine weighting:
$$g_1(u,v,\beta) = g(u,v,\beta)\frac{R_d}{\sqrt{R_d^2+u^2+v^2}} \tag{3}$$
2. ramp filtering:
$$g_2(u,v,\beta) = g_1(u,v,\beta) \otimes h_{ramp}(u) \tag{4}$$
3. backprojection:
$$\hat{f}(x,y,z) = \frac{1}{2}\int_0^{2\pi}(\frac{R_s}{R_s - xsin\beta + ycos\beta})^2 g_2(u,v,\beta)d\beta \tag{5}$$

Typically, the algorithm input data consist of K-projections, the size of each projection is $N_u \times N_v$, the output data is N^3-voxel volume V.

3 Materials and Methods

The CUDA programming model assumes that both the host (CPU) and the device (GPU) maintain their own separate memory spaces, referred to as host memory and device memory, respectively. Since a CUDA kernel works only on data present in device memory and due to the limited GPU memory capacity, it is not easy to store the entire volume and the projections in device memory. For example: a 512^3 voxel volume requires al least 512 MB of memory space, so we have decided to store the entire volume (or a 512^3 portion if the volume size is greater than 512^3) in device memory and to load the projections into device memory when needed and remove it after its backprojection. Our first implementation of FDK algorithm

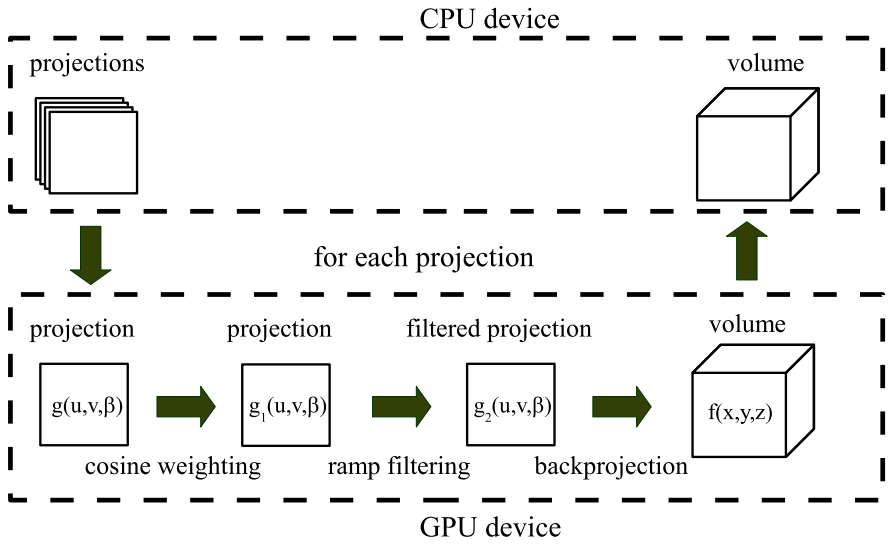

Figure 2: Overview of implemented naive method.

on CUDA-enabled GPU is a naive method, as shown in fig. 2, we transfer the first projection

P_1 to the device global memory and perform the cosine weighting, the ramp filtering and the backprojection steps to the volume V in the device global memory. This operation is repeated on the remaining projections to obtain the final accumulated volume. The naive code assigns in the backprojection step every voxel to a different CUDA threads. Two data sets have been chosen for testing the naive implementation in CUDA:

- **dataset 1** consists of 200 projections acquired on 360° circular scan trajectory. The size of each projection is 1024x512.

- **dataset 2** consists of 642 projections acquired on 360° circular scan trajectory. The size of each projection is 256x192.

Reconstruction tests have been performed on a desktop PC: it is equipped with a Intel Core i7-3770 CPU, 16 GB main memory and a nVIDIA GeForce GTX-Titan with 4GB device memory. Our implementation runs on Linux with CUDA v5.5. The filtering step has been implemented by using the CUFFT library of CUDA package. Table 1 shows the execution time needed for the CUDA implementation of FDK algorithm compared with the OpenMP [10] implementation.

	step1 [s]	step2 [s]	step3 [s]	Total [s]
CUDA				
dataset1	0.57	5.69	8.05	14.31
dataset2	0.45	1.81	3.1	5.36
OpenMP				
dataset1	0.67	5.09	206.95	212.71
dataset2	0.21	1.26	80.96	82.43

Table 1: Execution time of CUDA implementation compared to OpenMP. step 1: cosine weighting, step 2: ramp filtering, step 3: backprojection.

The results obtained show a remarkable time speed-up in the backprojection step implemented in CUDA by factor ≈ 25 respect to the OpenMP implementation, while the other ones are comparable. Starting from these results we have worked on the optimization of the backprojection step, a memory intensive operation, by using the techniques suggested in the CUDA programming guide [2]. The two techniques used are:

1. **Memory coalescing**: we organize the threads inside a block to access contiguous memory location and each thread computes the backprojection of a given number of voxels along z. This approach allows to save the number of add-multiply operations by incrementing the (u,v) coordinates.

2. **Global memory access reduction**: we use kinds of memory that have a cache mechanism (texture and constant memory). Moreover, texture memory has a GPU's hardware bilinear interpolation mechanism that speeds up the calculation of update voxel value. Also, we increase the number of projections processed by a single thread to reduce the number of access to volume global memory.

We have accomplished the task of backprojection optimization on CUDA-enabled GPU by using RabbitCT [11] an open platform for worldwide comparison in backprojection performance. We have implemented two new versions for the backprojection kernel, the version v1 where we

introduce the use of constant and texture memory on GPU, and the version v2 where in addition we load more projections in texture memory before the backprojection step. A processed dataset of a rabbit, suitable for cone beam 3D reconstruction, is available for testing the own kernel implementation. It consist of N=496 projection images I_n, the size of a projection image is 1248x960 pixels. The performance results of the new kernels respect to the naive kernel, named v0, are shown in table 2. Both the new versions of backprojection kernel show an improvement

Version	Size	Time [s]	Occupancy	Error [HU]	GUPs
v0	512	10.32	84	0.03	6.01
v1	512	4.77	89	0.16	12.90
v2	512	3.0	95	0.16	20.67

Table 2: Comparison of execution time, percentage of GPU occupancy , reconstruction error in Hounsfield Unit (HU), billion of voxel updates per second (GUPs), for the various version of backprojection kernel on the RabbiCT dataset.

in the execution time, the speed-up of version v1 is of a factor 2 and the speed-up of version v2 is of a factor 3 with a moderate increase of reconstruction error. At the end of optimization step the new backprojection kernels have been integrated in CBCT reconstruction code.

4 Results and Discussion

The reconstructed images of dataset 1 obtained by using CUDA and OpenMP implementation of the FDK method are shown in figure 3. The maximum relative difference between the

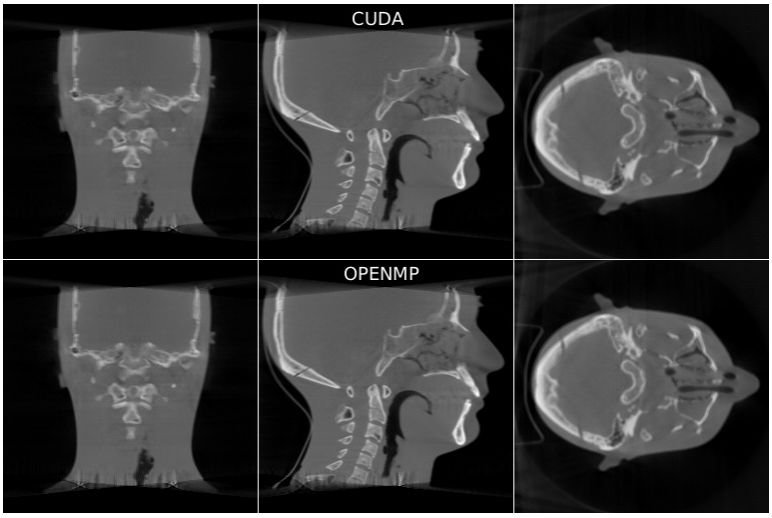

Figure 3: Comparison between GPU and CPU reconstructed CBCT images of dataset 1.

corresponding pixels in the reconstructed volumes, defined in eq. 6, is less than 9.0×10^{-3}.

$$\max_{(x,y,z)\in V} \frac{|I_{CUDA}(x,y,z) - I_{OMP}(x,y,z)|}{I_{OMP}(x,y,z} \tag{6}$$

The total execution times of various CUDA implementation of FDK reconstruction software are in table 3 compared to the OpenMP implementation.

	v0 [s]	v1 [s]	v2 [s]	OpenMP [s]
dataset1	14.31	8.49	7.68	212.7

Table 3: Total execution time of various versions of CUDA implementation of the FDK reconstruction algorithm compared with the OpenMP implementation.

Using GPU + CUDA in CBCT reconstruction we are able to accelerate the reconstruction process of a factor greater than 25. On dataset 1 the reconstruction time is less than 8 s for a volume of 512^3, and this result suggests CUDA implementation permits a real time reconstruction of CBCT data.

Acknowledgment

The GAP project is partially supported by MIUR under grant RBFR12JF2Z "Futuro in ricerca 2012".

References

[1] G. Pratx and L. Xing, Med. Phys. **38** 2685 (2011).

[2] nVIDIA Corporation: CUDA C Programming guide Version v5.5 (July 2013).

[3] F. Xu, K. Mueller, Phys. Med. Biol. **52** 3405 (2007).

[4] H. Scherl, B. Keck, M. Kowarschik, J. Hornegger, IEEE NSS Conf. Proc. **6** 4464 (2007).

[5] G. Yan, J. Tian, S. Zhu, Y. Dai, C. Qin, J. X-ray Sci. **16** 225 (2008).

[6] Y. Okitsu, F. Ino, K. Hagihara, Parallel Comput. **36** 129 (2010).

[7] M. Kachelrieβ, M. Knaup, O. Bockenbach, Med. Phys. **34** 1474 (2007).

[8] N. Gac, S. Mancini, M. Desvignes, Proc. 21st ACM Symp. Applied Computing 222 (2006).

[9] L.A. Feldkamp, L.C. Davis, J.W. Kress, J. Opt. Soc. Am. **A1** 612 (1984).

[10] B. Chapman, G. Jost, R. van der Pas, *Using OpenMP*. The MIT Press (2008).

[11] C. Rohkohl, B Keck, H.G. Hofmann,J. Hornegger, Med. Phys. **36** 3940 (2009).