Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA)

Holger Scherl, Benjamin Keck, Markus Kowarschik, and Joachim Hornegger

***Abstract*—The Common Unified Device Architecture (CUDA) is a fundamentally new programming approach making use of the unified shader design of the most current Graphics Processing Units (GPUs) from NVIDIA. The programming interface allows to implement an algorithm using standard C language and a few extensions without any knowledge about graphics programming using OpenGL, DirectX, and shading languages.**

We apply this revolutionary new technology to the FDK method, which solves the three-dimensional reconstruction task in cone-beam CT. The computational complexity of this algorithm prohibits its use for many medical applications without hardware acceleration. Today's GPUs with their high level of parallelism are cost-efficient processors for performing the FDK reconstruction according to medical requirements.

In this paper, we present an innovative implementation of the most time-consuming parts of the FDK algorithm: filtering and back-projection. We also explain the required transformations to parallelize the algorithm for the CUDA architecture. Our implementation approach further allows to do an on-the-fly-reconstruction, which means that the reconstruction is completed right after the end of data acquisition. This enables us to present the reconstructed volume to the physician in real-time, immediately after the last projection image has been acquired by the scanning device.

Finally, we compare our results to our highly optimized FDK implementation on the Cell Broadband Engine Architecture (CBEA), both with respect to reconstruction speed and implementation effort.

I. Introduction

The FDK method is used in most of today's cone-beam CT scanners as the standard reconstruction approach. The typical medical workflow requires high-speed reconstructions in order to avoid an interruption of patient treatment during surgery. Current GPUs offer massively parallel processing capability that can handle the computational complexity of three-dimensional cone-beam reconstruction. In comparison to the nine-way coherent Cell Broadband Engine Architecture, for which we have previously demonstrated a substantial reconstruction speed [1], the NVIDIA GeForce 8800 GTX architecture uses 128 Stream Processors in parallel. This GPU (345.6 Gflops[1], 128 stream processors, 1.35 GHz, one multiply-add operation per clock cycle per Stream Processor) is theoretically capable of sustaining almost twice the performance of the CBEA (208.4 Gflops, eight processing elements (SPE), 3.2 GHz, four multiply-add operations per clock cycle per SPE). Due to the fact that NVIDIA develops a fundamentally new easy-to-use computing architecture for solving complex computational problems on the GPU (CUDA), this speed-up factor should influence the high-speed reconstruction performance significantly.

CUDA offers a unified hardware and software solution for parallel computing on CUDA-enabled NVIDIA GPUs supporting the standard C programming language together with high performance computing numerical libraries[2].

This unveils the access to the processing power of graphics cards even for programmers that are not specialists in computer graphics. The implementation of the reconstruction task can now be done without knowing every trick how to (ab)use the existing Application Programming Interfaces (API), e.g. OpenGL, DirectX, or the Brook language, for general-purpose programming.

In this paper we detail the necessary steps for unveiling the reconstruction performance using this programming paradigm for commodity graphics hardware.

II. Related Work

Up to now, the reconstruction performance on graphics accelerators was evaluated only using graphics-based implementation approaches using OpenGL and shading languages [2], [3]. In comparison to the traditional graphics-based implementation methods, our CUDA-based implementation of cone-beam reconstruction has even a slightly improved reconstruction speed.

Despite of our optimized Cell-based back-projection implementation [1], another group also demonstrated cone-beam back-projection on the Cell processor [4]. The results are comparable to the performance of our back-projection module. The most time consuming operation in the inner loop of reconstruction is the ratio computation due to the perspective projection model. Our Cell-based approach avoids image rectification as suggested [4] that leads to the elimination of the homogeneous division [5] but introduces an additional low-pass operation on the projection.

III. Method

The FDK method can be divided into three steps: generate weighted projection data (cosine weighting and short-scan weighting), ramp filter the projection images row-wise, and

H. Scherl, B. Keck and J. Hornegger are with the Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Computer Science, Institute of Pattern Recognition (LME), Martensstr. 3, D-91058 Erlangen, Germany.

M. Kowarschik is with Siemens Medical Solutions, CO Division, Medical Electronics, Imaging, and IT Solutions, P.O.Box 3260, D-91050 Erlangen, Germany.

[1] 1 Gflops = 1 Giga floating point operations per second

[2] http://developer.nvidia.com/object/cuda.html

back-project the filtered projection data into the volume [6]. Because of deviations due to mechanical inaccuracies of real cone-beam CT systems such as C-arm scanners, the mapping between volume and projection image positions is described by a 3×4 projection matrix P_i for each X-ray source position i along the trajectory. In a calibration step the projection matrices are determined when the C-arm device is installed [7]. In this paper we concentrate on the two most compute-intensive tasks: filtering and back-projection.

IV. Implementation

Using the *Reconstruction Toolkit* (RTK) [8], we have implemented the basic processing chain as a pipeline architecture. One pipeline stage is responsible for loading the projections form the hard disk or over the network. As soon as a projection is available, the subsequent pipeline stages perform the filtering and back-projection, respectively. The pipeline stages for filtering and back-projection share the same thread of execution in order to make use of the CUDA-enabled GPU.

First, the projection image has to be transferred to the device memory of the graphics card. We used the *CUFFT* library of the CUDA package to implement the convolution with the given filter kernel. This library only supports the calculation of complex fast Fourier transforms (FFT). Therefore, we simultaneously convolve two image rows of a projection with the given filter kernel, where one image row defines the real input and the other one the imaginary input. The convolution is computed by a complex 1-D FFT followed by the point-wise multiplication of the discrete Fourier transform (DFT) of the filter kernel and the computation of the inverse FFT of the respective point-wise product. The necessary computations are mapped to several successive CUDA kernel executions: Data rearrangement to complex format, zero-padding, batched FFT, multiplication with the DFT of the filter kernel, batched IFFT, and data rearrangement to the original format. Each CUDA kernel computes all rows of a complete projection image simultaneously in order to make efficient use of the multiprocessors on the graphics card.

In the next step we perform the voxel-based back-projection, which requires the calculation of a matrix-vector product for each voxel in order to determine the corresponding projection value. Then we invoke our back-projection kernel on the graphics device. Each thread of the kernel computes the back-projection of a certain column and volume slice (see Figure 1). This allows to save six multiply-add operations by incrementing the homogeneous coordinates with the appropriate column of P_i for neighboring voxels in y-direction. This approach also reduces the register usage (see Figure 2).

In this regard, a rectification-based approach has theoretically the potential to further improve the reconstruction speed [5]. During our experiments we, however, observed that this is not true for the used graphics hardware. This optimization method, actually, degrades the reconstruction speed when compared to the approach described above.

Finally, we experimentally chose an efficient grid configuration (see Figure 3).

After all projections have been processed the volume is transferred to the host system memory. Our projection-based

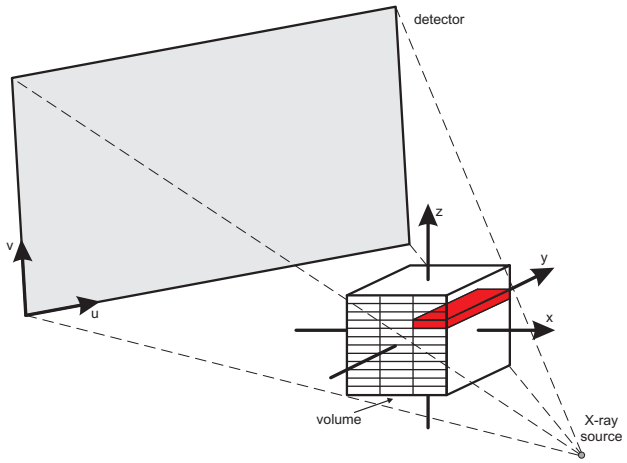

Fig. 1. Perspective geometry of the C-arm device (the v-axis and z-axis are not necessarily parallel) together with the parallelization strategy of our back-projection implementation on the GPU using CUDA (the x-z plane is divided in several blocks to specify a grid configuration, and each thread of a corresponding block processes all voxels in y-direction).

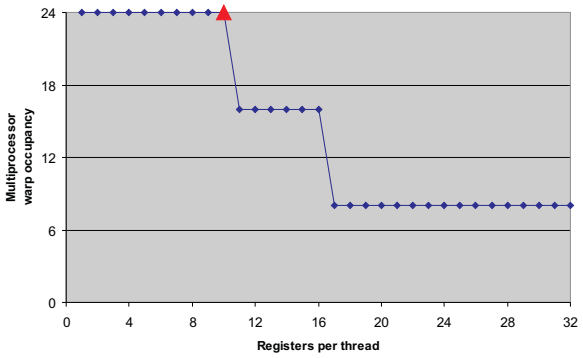

Fig. 2. Dependency of the multiprocessor warp occupancy on the register usage. Our CUDA implementation uses only 10 registers.

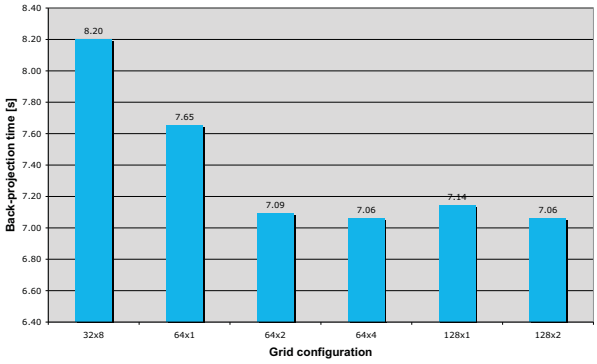

Fig. 3. Execution time for different grid configurations.

	Time [s]	pps	fps
Filtering			
NVIDIA GeForce 8800 GTX (CUDA)	3.00	138.00	
Cell processor 3.2 GHz (CBEA)	0.82	503.03	
Back-projection			
NVIDIA GeForce 8800 GTX (CUDA, NN/LI)	7.06	58.64	72.52
Cell processor 3.2 GHz (CBEA, NN)	11.85	34.94	43.21
Cell processor 3.2 GHz (CBEA, LI)	20.99	19.73	24.40
Data transfer (load projections / store volume)			
NVIDIA GeForce 8800 GTX (CUDA)	1.07 / 0.89		
Cell processor 3.2 GHz (CBEA)	0.00 / 0.00		
Overall execution (filtering, back-projection and data transfer)			
NVIDIA GeForce 8800 GTX (CUDA, NN/LI)	12.02	34.44	42.60
Cell processor 3.2 GHz (CBEA, NN)	13.60	30.44	37.64
Cell processor 3.2 GHz (CBEA, LI)	24.04	17.22	21.30

TABLE I
PERFORMANCE RESULTS OF FILTERING AND BACK-PROJECTION IN NEAREST NEIGHBOR INTERPOLATION (NN) AND BI-LINEAR (LI) INTERPOLATION MODE.

approach allows to do an on-the-fly reconstruction while projection data are still being acquired.

V. RESULTS

The filtering and back-projection code has been executed both on an Intel Xeon processor running at 2.8 GHz and equipped with an NVIDIA GeForce 8800 GTX graphics card, and on a Cell Blade server board with two Cell processors running at 3.2 GHz each and 1 GB of main memory split across the two chips. Only one Cell processor has been used during our measurements. The performance has been evaluated using a data set consisting of 414 projections of 1024×1024 pixels each and using a volume consisting of $512 \times 512 \times 512$ voxels. We have chosen a voxel size of 0.26^3 mm^3 in order to ensure that all voxels lie inside the field-of-view. In Table I we show the timing measurements for the filtering, back-projection, and also for the overall execution. We also give the numbers of projections that can be back-projected per second (pps) and the numbers of 512×512 volume slices, that can be reconstructed in one second (fps = frames per second).

As far as the filtering step is concerned, the Cell processor outperforms the GPU significantly. Although we assume that in this early CUDA release the FFT performance is not yet optimal, the Cell architecture could be used more efficiently, involving a very time-consuming hand optimization. Looking at the back-projection, the GPU clearly benefits from its theoretical advantage in processing power over the Cell processor. In addition, on the GPU the bilinear interpolation is nearly for free due to additional hardwired circuits on the device for fast texture access. The measured overall execution time allows to compute the FDK reconstruction on-the-fly ($>$30 fps) with either one GPU or two Cell processors.

VI. CONCLUSIONS

We have presented an optimized CUDA-based implementation of the FDK method. The resulting performance leverages cone-beam CT reconstructions on-the-fly, which means that we can hide all required computations behind the scan-time of the used acquisition device. Due to the exclusive use of off-the-shelf hardware components the physician can now run retrospective reconstructions with varying reconstruction parameters, e.g. different filter kernels, using a standard PC.

In contrast to traditional implementation approaches using OpenGL and shading languages, for example, the CUDA architecture enables the ”non-graphics” programmer to implement efficient GPU code for general-purpose computations in a more simplified and appropriate way. With respect to the implementation of the FDK algorithm the CUDA-based approach required much less implementation effort than an optimized CBEA-based implementation.

We conclude that the CUDA-enabled GPU is well suited for delivering high-speed reconstructions in flat-panel cone-beam CT (e.g., C-arm devices).

ACKNOWLEDGMENTS

This work was supported by Siemens Medical Solutions, CO Division, Medical Electronics, Imaging, and IT Solutions.

REFERENCES

[1] H. Scherl, M. Koerner, H. Hofmann, W. Eckert, M. Kowarschik, and J. Hornegger, “Implementation of the FDK algorithm for cone-beam CT on the Cell Broadband Engine Architecture,” in *Proceedings of SPIE Medical Imaging 2007: Physics of Medical Imaging*, J. Hsieh and M. Flynn, Eds., vol. 6510, San Diego, February 2007, p. 651058.

[2] K. Mueller, F. Xu, and N. Neophytou, “Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?” in *SPIE Electronic Imaging Conference*, San Diego, 2007, (Keynote, Computational Imaging V).

[3] M. Churchill, “Hardware-accelerated cone-beam reconstruction on a mobile C-arm,” in *Proceedings of SPIE*, J. Hsieh and M. Flynn, Eds., vol. 6510, 2007.

[4] M. Kachelrieß, M. Knaup, and O. Bockenbach, “Hyperfast perspective cone-beam backprojection,” in *IEEE Nuclear Science Symposium and Medical Imaging Conference*, San Diego, 2006, m01-7.

[5] C. Riddell and Y. Trousset, “Rectification for cone-beam projection and backprojection,” *IEEE Transactions on Medical Imaging*, vol. 25, no. 7, pp. 950–962, 2006.

[6] H. Turbell, “Cone-beam reconstruction using filtered backprojection,” Ph.D. dissertation, Linköping University, Sweden, SE-581 83 Linköping, Sweden, February 2001, dissertation No. 672, ISBN 91-7219-919-9.

[7] K. Wiesent, K. Barth, N. Navab, P. Durlak, T. Brunner, O. Schuetz, and W. Seissler, “Enhanced 3-D-reconstruction algorithm for C-arm systems suitable for interventional procedures,” *IEEE Transactions on Medical Imaging*, vol. 19, no. 5, pp. 391–403, 2000.

[8] H. Scherl, S. Hoppe, M. Kowarschik, and J. Hornegger, “Design and implementation of the software architecture for a 3-D reconstruction system in medical imaging,” Leipzig, 2008, submitted to IEEE International Conference on Software Engineering.