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

Holger Scherl[1], Benjamin Keck[1], Markus Kowarschik[2], Joachim Hornegger[1]

SIEMENS

[1] Institute of Pattern Recognition (LME), Universität Erlangen-Nürnberg, Martensstr. 3, D-91058 Erlangen, Germany; {scherl, keck, hornegger}@informatik.uni-erlangen.de

[2] Siemens AG, Medical Solutions, CO Division, Medical Electronics, Imaging, and IT Solutions, P.O.Box 3260, D-91050 Erlangen, Germany; markus.kowarschik@siemens.com

Motivation

- FDK method [1] is the state-of-the-art reconstruction approach in cone-beam CT
- High computational complexity
- Prohibits its use for many medical applications without hardware acceleration
- Modern GPUs offer an immense computing power (>345 Gflops)
- Innovative implementation of the FDK algorithm on graphics accelerator hardware from NVIDIA using CUDA

Method

The FDK method can be divided into three steps:

- Generate weighted projection data (cosine weighting)
- Ramp-filter the projections row-wise
- Back-project the filtered projection data into the volume

For short-scan reconstructions sinogram weighting, e.g. Parker weighting, has to be applied.

Implementation

- Pipelined approach
- *Reconstruction Toolkit* (RTK) [2]
- Pipeline stages for filtering and back-projection share the same thread of execution

Filtering

- FFT-based convolution
- Usage of the *CUFFT* library from NVIDIA
- Process several projection rows simultaneously in one kernel
- Supports different convolution lengths

Back-Projection

- Voxel-based back-projection [3]
- Use of projection matrices in order to deal with non-ideal geometry
- Choose a grid configuration (Figure 2)
- Incremental implementation approach in y-direction
- Optimize register usage (Figure 3)

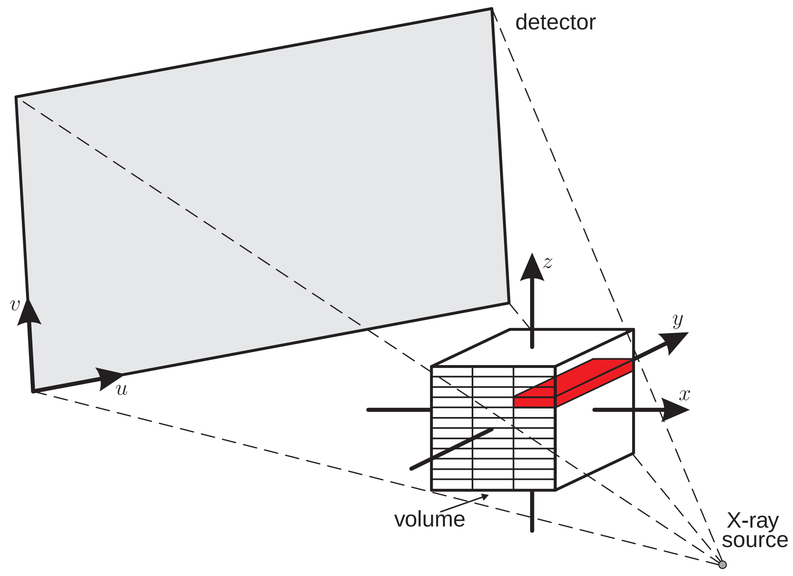

Figure 1: Perspective geometry of the C-arm device (the v-axis and z-axis are not necessarily parallel) together with the parallelization strategy of our back-projection implementation on the GPU using CUDA (the x-z plane is divided in several blocks to specify a grid configuration, and each thread of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

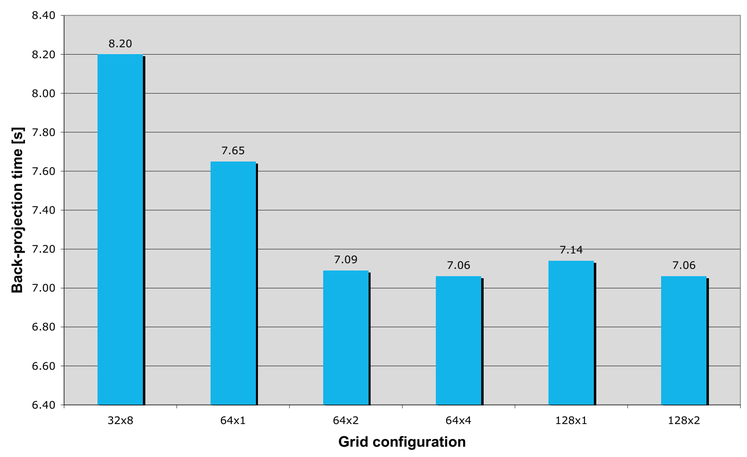

Figure 2: Execution time for different grid configurations.

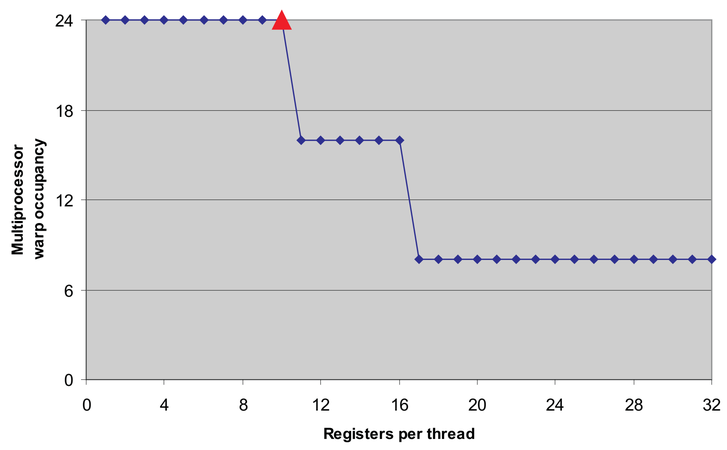

Figure 3: Dependency of the multiprocessor warp occupancy on the register usage[1]. Our CUDA implementation uses only 10 registers.

Experimental Setup

- Host system equipped with Intel Xeon processor (2.8 GHz, 4 GB main memory)
- NVIDIA GeForce 8800 GTX GPU (345.6 Gflops[2], 128 stream processors, 1.35 GHz, one multiply-add operation per clock cycle per Stream processor)
- Best runtime out of five measurements
- Dataset consists of 414 projection images
- Projection size is 1024×1024 pixels
- Volume dimension is 512^3 voxels
- Volume geometry chosen such that all voxels are contained inside the FOV

Results

- Execution time using our CUDA implementation on an NVIDIA GeForce 8800 GTX board
- Bilinear interpolation mode is as fast as nearest neighbor interpolation because of special hardware support on the GPU
- Comparison to our optimized Cell implementation [4]
- Most important for on-the-fly-reconstruction is the number of processed projections per second (pps)

	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 1: Execution time of filtering and back-projection using nearest neighbor (NN) and bilinear interpolation (LI).

Conclusions

- Highly optimized implementation on NVIDIA GPUs using CUDA
- Twice as fast compared to our optimized implementation on the Cell processor [4]
- All required computations are hidden behind the scan-time of the used C-arm device
- On-the-fly-reconstruction

Acknowledgments

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

References

[1] L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone-beam algorithm. *J. Opt. Soc. Amer.*, A1(6):612–619, 1984.

[2] 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, 2008. submitted to IEEE International Conference on Software Engineering.

[3] 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*, 19(5):391–403, 2000.

[4] 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 J. Hsieh and M. Flynn, editors, *Proceedings of SPIE*, volume 6510, San Diego, February 2007.

[1] Figure created using the CUDA Occupancy Calculator v1.2 (.xls) from Nvidia

[2] 1 Gflops = 1 Giga floating point operations per second