Comparing performance of many-core CPUs and GPUs for static and motion compensated reconstruction of C-arm CT data

Hannes G. Hofmann,[a] Benjamin Keck, and Christopher Rohkohl
Pattern Recognition Laboratory, Universität Erlangen-Nürnberg, 91058 Erlangen, Germany

Joachim Hornegger
Pattern Recognition Laboratory, Universität Erlangen-Nürnberg, 91058 Erlangen, Germany
and Erlangen Graduate School in Advanced Optical Technologies (SAOT), 91058 Erlangen, Germany

(Received 4 June 2010; revised 12 November 2010; accepted for publication 16 November 2010; published 29 December 2010)

Purpose: Interventional reconstruction of 3-D volumetric data from C-arm CT projections is a computationally demanding task. Hardware optimization is not an option but mandatory for interventional image processing and, in particular, for image reconstruction due to the high demands on performance. Several groups have published fast analytical 3-D reconstruction on highly parallel hardware such as GPUs to mitigate this issue. The authors show that the performance of modern CPU-based systems is in the same order as current GPUs for static 3-D reconstruction and outperforms them for a recent motion compensated (3-D+time) image reconstruction algorithm.
Methods: This work investigates two algorithms: Static 3-D reconstruction as well as a recent motion compensated algorithm. The evaluation was performed using a standardized reconstruction benchmark, RABBITCT, to get comparable results and two additional clinical data sets.
Results: The authors demonstrate for a parametric B-spline motion estimation scheme that the derivative computation, which requires many write operations to memory, performs poorly on the GPU and can highly benefit from modern CPU architectures with large caches. Moreover, on a 32-core Intel® Xeon® server system, the authors achieve linear scaling with the number of cores used and reconstruction times almost in the same range as current GPUs.
Conclusions: Algorithmic innovations in the field of motion compensated image reconstruction may lead to a shift back to CPUs in the future. For analytical 3-D reconstruction, the authors show that the gap between GPUs and CPUs became smaller. It can be performed in less than 20 s (on-the-fly) using a 32-core server.
[DOI: 10.1118/1.3525838]

Key words: GPU, multi-core, CBCT, recon, high-performance

I. INTRODUCTION

Recent C-arm systems with flat panel detectors allow the rotational acquisition of high-resolution projection images for 3-D volume reconstruction.[1] The additional intraprocedural 3-D information enables advanced techniques for diagnosis (e.g., perfusion imaging), planning, navigation, treatment, and follow-up. Recently, the reconstruction of moving structures has also become a matter of research. Time resolved image data can support complex interventional procedures in cardiology, such as transcutaneous valve replacement, implantation of biventricular pacemakers, and the assessment of myocardial perfusion.[2]

For seamless integration into clinical workflows, it is highly desirable to introduce as little time penalty as possible. Therefore, the 3-D reconstruction has to be performed concurrently with the acquisition of the projection images (on-the-fly), finishing just after the acquisition. Table I lists common clinical protocols that show the targeted time range. In order to fulfill these real-time requirements, hardware acceleration is mandatory. Of course, the real-time requirement is valid for the complete image processing chain from image acquisition over preprocessing to 3-D reconstruction. However, since the reconstruction step is the major bottleneck, we focus on that part in this work.

Hardware acceleration has been an issue of high importance in the recent past in all areas of medical image computing, e.g., registration,[3] segmentation,[4] and reconstruction.[5] The decision for the most suitable hardware platform is domain-specific and depends on the amount of data, the computational load, and the parallelism in the algorithm. Standard reconstruction algorithms are embarrassingly parallel and hence can exploit the high level of parallelism provided by modern hardware, such as the vector processing units of current CPUs, the eight processing elements of IBM's CELL processor,[6] or the many shader cores of modern graphics processing units (GPUs). For 3-D reconstruction, recent publications have demonstrated that GPUs have quite a speed advantage over CPUs for standard algorithms.[7] However, algorithmic innovations in the field of motion compensated (3-D+time) image reconstruction[8–10] may lead to a shift back to CPUs in the future. The reduced degree of parallelism in these novel methods results from various issues, e.g., irregular memory access patterns or data dependencies.

TABLE I. Common C-arm CT scan protocols, according to Strobel *et al.* (Ref. 1) (HC: High-contrast; LC: Low-contrast).

Name	Binning	Matrix size (px)	No. frames	f s^{-1} (s^{-1})	Scan time (s)
Head (HC)	2×2	1240×960	133	30	5
Body (HC)	2×2	1240×960	133	30	5
Head (LC)	2×2	1240×960	496	30	20
Body (LC)	4×4	620×480	397	60	8

The main contribution of this paper is twofold. First, we show that the gap between GPUs and CPUs becomes smaller for static 3-D reconstruction due to CPU evolution. Second, we demonstrate that novel more complex algorithms can highly benefit from modern CPU architectures with large caches.

II. METHODS AND MATERIALS

II.A. Medical image computing architectures: CPU vs GPU

Computing performance is affected by several factors. Hardware performance depends highly on the number of computing elements, its clock speed, and the memory hierarchy. On the other hand, the performance of a specific algorithm depends, e.g., on the ratio of read and write operations or the ratio of memory accesses and compute operations. For comparing high-end CPUs and GPUs we examine two current GPUs from NVIDIA® and three different Intel® Xeon® servers (cf. Table II). In the following, we give a brief overview of both architectures and the implications for algorithmic performance.

CPUs have always been general purpose processors able to execute arbitrary computing tasks. Over time, parallel execution units were added. Current CPUs are able to perform a single instruction on four data elements simultaneously [single instruction, multiple data (SIMD)] with their vector units and feature up to eight cores on a single chip, with four chips in a PC that makes 32 cores in a single unit. CPUs feature a cache hierarchy of three levels of increasing size and access to the huge main memory. Cache sizes are up to 16 MB on a single chip and main memory can comprise hundreds of GB.

GPUs were developed for one specific task: Rasterization. Only recently have they become usable for generic computations, thanks to their programmable shaders. Current GPUs feature up to 448 stream processors corresponding to the individual lanes of traditional SIMD units. GPUs provide shared memory and read-only caches for texture memory and constant data. Shared memory is fast but relatively small; 8/16 processing units share only 16/48 KB (GT200/Fermi architecture). The GPU chip is located on an extension board

Algorithm 1: Basic steps for back-projection of image $\boldsymbol{I}_i$.

```
foreach voxel x do
  project x onto detector plane;
  (pre-)fetch 4 projection values;
  wait for projection values;
  bilinear interpolation;
  update f_FDK(x);
end
```

Algorithm 2: Basic steps of 4-D reconstruction of image $\boldsymbol{I}_i$.

```
foreach voxel x do
  compute motion vector→x';
  project x' onto detector plane;
  (pre-)fetch 4 projection values;
  wait for projection values;
  bilinear interpolation;
  compute gradients of projection image;
  for z=0 to 3 do
    for y=0 to 3 do
      for x=0 to 3 do
        for t=0 to 3 do
          update motion field gradients;
        end
      end
    end
  end
  update f_FDK(x);
end
```

TABLE II. Main characteristics of the five computer systems used. For the Tesla™ systems, numbers for the host (CPU) system are omitted.

	Xeon®5500	Xeon®X7460	Xeon®32-core	Tesla™C1060	Tesla™C2050
Cores/shaders	2∗4=8	4∗6=24	4∗8=32	240	448
Clock (GHz)	2.66	2.66	2.27	1.3	1.15
Microarchitecture	Nehalem	Core	Nehalem	GT200	Fermi
RAM Size (GB)	12	32	64	4	3
RAM type	DDR3–1066	DDR2–1066	DDR3–1066	GDDR3	GDDR5

equipped with its own memory, introducing an extra level in the memory hierarchy. The GPU's memory has a very high bandwidth but is smaller than main memory (up to 6 GB).

Medical image processing algorithms consist of several common patterns which benefit differently from each platform. Data interpolation is a part of most algorithms, e.g., in registration, reconstruction, and segmentation. GPUs with their dedicated texture sampling hardware can greatly accelerate this operation. A prominent example is ray-casting with trilinear interpolation.[11] Typical medical data sets comprise a huge amount of data. The high memory bandwidth of GPUs is beneficial if the size is sufficient. However, the data have to be transferred from the main memory to the device and back. CPUs benefit from their huge caches, allowing fast read and write access to data from main memory.

II.B. Impact of CPU evolution on static 3-D reconstruction

Today's reconstruction implementations are usually based on the FDK method.[12] We focused on the backprojection and measured the performance of already preprocessed data sets. This preprocessing includes physical corrections (beam hardening and scatter), redundancy weighting, and filtering according to Strobel *et al.*[1] As reported in literature[13] it makes up only about 10% of the total runtime.

Algorithm 1 shows the basic steps of the backprojection of one projection image. For each projection image, all voxels $\underline{x}=(x,y,z)$ are updated with the corresponding detector value which is determined by perspective projection of the voxel onto the detector. The FDK method is embarrassingly parallel and hence can exploit the high level of parallelism provided by modern hardware. On GPUs, it benefits from hardware-supported interpolation of projection images and the high memory bandwidth. However, several aspects of CPU development have increased their performance to a similar level. The number of cores was raised to 32 in one system, which, with four times SIMD, equal 128 GPU processing units. Current CPUs feature an increased memory bandwidth due to their integrated memory controllers and faster RAM technology.

We applied various optimization techniques to the FDK algorithm to exploit the parallel processing capabilities of current CPUs and GPUs. They are detailed below.

II.B.1. CPU-optimization techniques

The backprojection problem is embarrassingly parallel since there are no data dependencies between the voxels. For each projection image all voxels can be updated independently from all others. Only concurrent updates of the same voxel from different viewing directions have to be avoided since this could cause incorrect results.

II.B.1.a. Multithreading and vectorization. Multithreading was implemented using Intel's Threading Building Blocks library by partitioning the reconstructed volume into many subvolumes that were processed by individual computing threads. Critical code parts, i.e., the inner loop, were manually vectorized using SIMD intrinsics.

II.B.1.b. Loop transformations. Loop transformations (loop unrolling, loop fusion) can reduce the number of branches in the code and enable the compiler to schedule instructions better, such that latencies are hidden and overall instruction throughput is improved.

II.B.1.c. Temporal blocking. Temporal blocking is a technique to improve the use of the fast caches. Voxels are updated several times before storing them back to main memory. Consequently, the bandwidth requirements of the algorithm are reduced. Since backprojection is bandwidth-limited on most platforms, temporal blocking can significantly reduce overall execution time.

II.B.1.d. Early termination. As mentioned before, subvolumes were used for multithreading. For each subvolume, the computation is aborted early if it is not in the field-of-view (FOV) of the current projection image. To check this, the eight corner voxels are projected into the detector plane. Then, the area of intersection between this "shadow" and the detector is calculated.

II.B.2. GPU-optimization techniques

Some of the CPU-optimization strategies (see Sec. II B 1) were also applied on the GPU. GPUs are able to switch threads fast and thereby hide memory latency. Therefore, the volume was partitioned into many chunks which are processed in parallel. The parallelization and optimization strategies are adapted from Scherl *et al.*[14] First of all, the hardware texture interpolation units were utilized for projection access. Further, memory accesses must be coalesced on GPUs to get good throughput. Therefore, the problem was reorganized as a 2-D problem, where every kernel loops over all voxels in the y-direction. Further details can be found in the original paper.[14]

II.C. Motion estimation for 3-D+time reconstruction

In a last year's MICCAI contribution,[9] a novel algorithm was presented that is able to estimate 4-D nonperiodic motion patterns using a time-continuous cubic B-spline motion model. The most time-consuming part of this method is the motion estimation as illustrated in Algorithm 2. Here, $\underline{x}$ is the (x,y,z) coordinate on the control grid, $\underline{x}'$ denotes the transformed position, and t refers to the time. In each iteration of the optimization procedure, the derivative of the FDK reconstruction formula with respect to the motion model parameters has to be computed. This computation is analogous to a FDK reconstruction. However, instead of accumulating voxel values in the innermost loop, the derivative of the B-spline tensor product is evaluated in four dimensions. Cubic B-splines depend on four sample points in each dimension. In total, this results in 256 (4^4) times more data to be stored. For the application of interest (cardiac vasculature reconstruction), only a sparse set of voxels has to be considered, ranging from 0.1% to 2%. However, for future applications and other algorithms, the number of voxels can be as high as 100%. A GPU implementation requires sparse sampling if high-resolution volumes are to be reconstructed. In general, GPUs have less memory than CPUs and therefore,

TABLE III. Description of the three data sets used.

Data set	Number of projections	Projection size
(A)	496	1240×960
(B)	543	1240×960
(C)	414	1024×1024

CPUs will be able to handle larger volumes. This can be nicely illustrated by the state-of-the-art Tesla™ C2070 GPU which is equipped with 6 GB onboard memory. For the presented problem, it allows only the derivative of a 184^3 dense volume to be completely stored in the device memory ($4\ \text{byte}*256*184^3=5.94\ \text{GB}$).

II.C.1. CPU-optimization techniques

Optimization was straightforward, analogous to the static case (cf. Sec. II B). The inner loop now contains the computation of the derivative for each voxel with respect to the B-spline model parameters. On the CPU, multithreading was implemented using OpenMP. The sparse set of voxels was distributed equally among all computing threads. The inner loops within the derivative computation were unrolled.

II.C.2. GPU-optimization techniques

The volume and the current projection are stored in the GPU device memory to avoid frequent data transfers over the PCIe bus. A sparse representation of the volume was used to reduce the memory requirements. The backprojection part was optimized analogous to the static case (cf. Sec. II B), e.g., use of texture units. In contrast, coalesced memory access cannot be used, since the offset of the memory access depends on the motion.

III. EXPERIMENTAL SETUP

III.A. Computer systems

All methods were benchmarked on three CPU-based many-core systems from Intel and on two NVIDIA® Tesla™ GPUs. The main specs of all four systems are summarized in Table II. The first CPU system comprises two Xeon® 5550 quad-core processors (eight cores total). The second one features four Xeon® X7460 hexacore processors (24 cores). The third one has four Xeon® eight-core processors (32 cores). Hyperthreading was enabled on the eight-core and 32-core systems, hence they exposed twice as many virtual processors to the operating system. Additionally to the CPU-based systems, we ran the tests on two workstations, each equipped with a NVIDIA® Tesla™ accelerator card (C1060 and C2050) with CUDA 3.1.

All tests on CPUs were performed using 64-bit Linux and the Intel compilers in version 11.1. All CPU-based systems used in this report are preproduction systems. Production hardware is expected to deliver similar performance levels.

III.B. Static 3-D reconstruction

The evaluation of our implementation of static 3-D reconstruction was performed using the open reconstruction benchmark RABBITCT.[15] It comprises a public data set and a test framework that allow for presentation of comparable results. The task of RABBITCT is to implement a standard FDK algorithm and reconstruct volumes of sizes 256^3, 512^3, or 1024^3 voxels from a standardized data set. Additionally to performance measurements, it allows easy verification of the implementation's correctness.

In this work, we reconstructed a volume size of 512^3 voxels. Three different data sets were used for evaluation and verification of the optimized static 3-D reconstruction (cf. Table III). Data set (A) is the publicly available RABBITCT data set and data sets (B) and (C) are clinical cases acquired with a Siemens C-arm system. Table III lists the number of projections and the dimensions of the projection images for all three data sets.

We evaluated three different optimization levels. The baseline was defined by our vectorized and multithreaded implementation[16] ("Base" in Table IV). "Cache" denotes a cache-optimized implementation with added temporal blocking. Finally, "E.term" is the most optimized version with temporal blocking and early termination. The subvolume size for Cache and E.term was empirically chosen to be $128\times64\times4$ as this configuration performed best over all architectures and data sets.

III.C. Motion estimation for 3-D+time reconstruction

The experiment for this paper was modeled after a clinical protocol used in Ref. 9. The B-spline motion model was parametrized by 5^3 control points in space and 35 in time resulting in a total of $3*5^3*35$ degrees of freedom. We used 133 projection images with a size of 960×960 pixels. The reconstructed volume had a size of 256^3 voxels. The sampling of the volume was set to 5%. The mean execution time

TABLE IV. Runtimes (in seconds) for the static reconstruction on four systems and three implementations ("Base," "Cache," and "E.term"). The volume size was 512^3 and the subvolume size $128\times64\times4$.

	Xeon® 5500			Xeon® X7460			32-core Xeon®				
Data set	Base	Cache	E.term	Base	Cache	E.term	Base	Cache	E.term	Tesla™ C1060	Tesla™ C1060
(A)	84.71	80.28	61.38	60.53	42.89	31.96	27.71	24.13	18.82	14.33	8.97
(B)	92.89	87.56	67.39	66.04	48.77	35.08	30.36	26.64	20.58	17.30	9.55
(C)	70.54	68.23	54.54	50.59	36.05	29.21	23.19	20.00	16.42	12.17	6.88

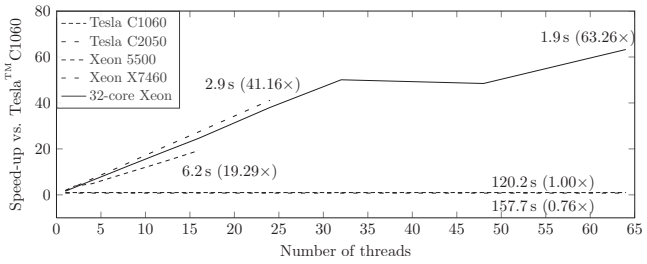

FIG. 1. Speed-up for motion estimation on the three Xeon® servers using a varying number of threads compared to the Tesla™ C1060. Note that the Tesla™ C2050 performed worse (0.76×).

for one iteration was measured by averaging 100 iterations. A typical reconstruction would require 50–100 iterations.

IV. RESULTS AND DISCUSSION

IV.A. Static 3-D reconstruction

Table IV shows the results for all four computer systems and for three different optimization methods. Depending on the system under investigation, the optimization techniques have varying impact. The effect of optimized cache usage due to temporal blocking correlates with the relative memory bandwidth available to each core. The eight-core Intel® Xeon® 5500 server has sufficient bandwidth and thus the speed-up was only up to 1.06×. On the 32-core Intel® Xeon® server, we observed a speed-up of at least 1.14×. Finally, the 24-core X7460 system is really bandwidth-starving, hence the good result of at least 1.35×. This observation demonstrates the impact of the higher memory bandwidth of the Nehalem microarchitecture. The proposed method of early termination further reduced the reconstruction time to about 72%–82% compared to the cache-optimized version on all three systems (1.22–1.39×). The effectiveness of the approach can be seen by looking at the numbers of a single data set, e.g., (A). With the chosen size of $128\times64\times4$ voxels, about 24% of the subvolumes are outside of the FOV. The reconstruction time is 75%–78% of the Cache implementation s on all systems. In the end, the newest GPU (Tesla™ C2050) was the fastest system, more than twice as fast as the best CPU system.

IV.B. Motion estimation for 3-D+time reconstruction

Figure 1 shows the results for the three CPU-based systems with varying numbers of threads. All three CPU systems performed better than the GPU. We observed an almost linear speed-up with the number of threads used. On Nehalem-based systems, the maximum speed-up was even higher than the physical number of cores due to hyperthreading. The 32-core Intel® Xeon® server outperformed the fastest GPU (C1060) by a factor of 63.26 when using 64 threads. Note that the C2050 performed worse than the C1060 because the drivers and compiler optimizations for the Fermi architecture are still under development. We further observed side effects of pinning threads to specific CPU cores, which is usually done to increase performance. When using only one or two threads on the Xeon® 5500 system, pinning resulted in plummeting performance. We suppose this to happen because the cores with threads pinned to them get too hot, disabling the built-in overclocking mechanism ("Turbo-Mode").

The relatively bad performance of the GPU may be explained by the frequent memory write-access operations. Due to its architecture (cf. Sec. II A), the CPU benefits from its fast caches which are not present on current GPUs. Further, the four nested inner loops of the derivative computation cause a high register pressure on the GPU. Consequently, less threads could be executed simultaneously, reducing the ability to hide memory latency.

V. CONCLUSION AND OUTLOOK

In this paper, we have shown two things. First, we have shown that static 3-D reconstruction can be computed on-the-fly on a high-end multi-CPU server, although GPUs are still faster. Our study only investigated the major part of the reconstruction, the backprojection, which accounts for more than 90% of the computation time. Regarding the real-time requirements in a realistic scenario, additional preprocessing and data transfer has to be performed. As an example, we consider the most demanding protocol from Table I [Head (LC)], with an acquisition time of 20 s. Preprocessing roughly adds 1–2 s to the 18.82 s on the CPU or 8.98 s on the C2050, respectively. Data transfer does not need to be considered as it can be hidden by streaming projections from the system to the workstation in the background. Further, we demonstrated that a more complex algorithm, in our example a motion compensated 3-D+time reconstruction, can highly benefit from the large caches of modern CPU architectures.

A Tesla™ C2050 GPU was 2.1 times faster for the static problem and the 32-core Intel® Xeon® server more than 60 times faster for the motion estimation part of the motion compensated algorithm. This illustrates nicely that for complex algorithms, a careful combination of both acceleration platforms can lead to an optimal result. This point is also reflected in the future hardware development of two major vendors, Intel® and NVIDIA®, which indicates that CPUs and GPUs are becoming more akin. Our results suggest that to achieve maximum performance, an expensive high-end server is necessary. However, even an off-the-shelf quad-core workstation is able to outperform a Tesla™ C1060 GPU by more than four times.

In summary, we could show that the ever increasing complexity of medical image computing algorithms requires paying close attention on the hardware architecture in order to enable their integration into the clinical workflow.

ACKNOWLEDGMENTS

Thanks to the Regional Computing Center Erlangen (RRZE) for support and hardware access. Thanks to the staff and facilities of the Intel® Manycore Testing Lab (http://software.intel.com/en-us/articles/intel-many-core-testing-lab/). This work was funded by a research grant from Intel.

[a]Electronic addresses: hannes.hofmann@cs.fau.de

[1]N. Strobel *et al.*, "3D imaging with flat-detector C-arm systems," in *Multislice CT*, 3rd ed. (Springer, Berlin, 2009), pp. 33–51.
[2]H. Hetterich, T. Redel, G. Lauritsch, C. Rohkohl, and J. Rieber, "New X-ray imaging modalities and their integration with intravascular imaging and interventions," The International Journal of Cardiovascular Imaging **26**(7), 797–808 (2010).
[3]J. Rohrer, L. Gong, and G. Szekely, in *Parallel Mutual Information Based 3D Non-Rigid Registration on a Multi-Core Platform*, Proceedings of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI) (New York, 2008).
[4]J. M. R. Byrd, S. A. Jarvis, and A. H. Bhalerao, in *Speculative Moves: Multithreading Markov Chain Monte Carlo Programs*, Proceedings of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI) (New York, 2008).
[5]P. B. Noël, A. Walczak, K. R. Hoffmann, J. Xu, J. J. Corso, and S. Schafer, in *Clinical Evaluation of GPU-Based Cone Beam Computed Tomography*, Proceedings of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI) (New York, 2008).
[6]M. Kachelrieß, M. Knaup, and O. Bockenbach, "Hyperfast parallel-beam and cone-beam back-projection using the CELL general purpose hardware," Med. Phys. **34**(4), 1474–1486 (2007).
[7]K. Mueller, F. Xu, and N. Neophytou, in *Why do Commodity Graphics Hardware Boards (GPUs) Work So Well for Acceleration of Computed Tomography?*, SPIE Electronic Imaging Conference, San Diego, Vol. 6498, 2007.
[8]E. Hansis, D. Schäfer, O. Dössel, and M. Grass, "Projection-based motion compensation for gated coronary artery reconstruction from rotational x-ray angiograms," Phys. Med. Biol. **53**(14), 3807–3820 (2008).
[9]C. Rohkohl, G. Lauritsch, M. Prümmer, and J. Hornegger, in *Interventional 4–D Motion Estimation and Reconstruction of Cardiac Vasculature Without Motion Periodicity Assumption*, Medical Image Computing and Computer-Assisted Intervention—MICCAI 2009, edited by G. Z. Yang, D. Hawkes, D. Rueckert, A. Noble, and C. Taylor (Springer, Berlin/Heidelberg, 2009); [Lect. Notes Comput. Sci. **5761**, 132–139 (2009)]
[10]A. Keil, J. Vogel, G. Lauritsch, and N. Navab, in *Dynamic Cone Beam Reconstruction using a New Level Set Formulation*, Medical Image Computing and Computer-Assisted Intervention—MICCAI 2009, edited by G.Z. Yang, D. Hawkes, D. Rueckert, A. Noble, and C. Taylor (Springer, Berlin/Heidelberg, 2009); [Lect. Notes Comput. Sci., **5762**, 389–397 (2009)].
[11]B. Keck, H. Hofmann, H. Scherl, M. Kowarschik, and J. Hornegger, in *GPU-Accelerated SART Reconstruction using the CUDA Programming Environment*, SPIE Medical Imaging Conference Proceedings, Lake Buena Vista, 7258, 72582B.1–72582B.9 (2009).
[12]L. Feldkamp, L. Davis, and J. Kress, "Practical cone-beam algorithm," J. Opt. Soc. Am. A **1**(6), 612–619 (1984).
[13]B. Heigl and M. Kowarschik, in *High-Speed reconstruction for C-Arm Computed Tomography*, Ninth International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine (Lindau, 2007), pp. 25–28.
[14]H. Scherl, B. Keck, M. Kowarschik, and J. Hornegger, in *Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA)*, Nuclear Science Symposium Conference Record, edited by E. C. Frey, Honolulu, Vol. 6, pp. 4464–4466, 2007.
[15]C. Rohkohl, B. Keck, H. G. Hofmann, and J. Hornegger, "CT—An open platform for benchmarking 3D cone-beam reconstruction algorithms," Med. Phys. **36**(9), 3940–3944 (2009).
[16]H. G. Hofmann, B. Keck, C. Rohkohl, and J. Hornegger, *PARS-Mitteilungen*, 26 (Gesellschaft für Informatik e.V., Bonn, 2009), pp. 91–100.