Implementation of the FDK Algorithm for Cone-Beam CT on the Cell Broadband Engine Architecture

Holger Scherl[a], Mario Koerner[a], Hannes Hofmann[a], Wieland Eckert[b], Markus Kowarschik[c], Joachim Hornegger[a]

[a]Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Computer Science, Institute of Pattern Recognition, Martensstr. 3, D-91058 Erlangen, Germany;
[b]Siemens Medical Solutions, AX Division, P.O.Box 1266, D-91294 Forchheim, Germany;
[c]Siemens Medical Solutions, CO Division, P.O.Box 3260, D-91050 Erlangen, Germany

ABSTRACT

In most of today's commercially available cone-beam CT scanners, the well known FDK method is used for solving the 3D reconstruction task. The computational complexity of this algorithm prohibits its use for many medical applications without hardware acceleration. The brand-new Cell Broadband Engine Architecture (CBEA) with its high level of parallelism is a cost-efficient processor for performing the FDK reconstruction according to the medical requirements. The programming scheme, however, is quite different to any standard personal computer hardware. 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 CBEA. Our software framework allows to compute the filtering and back-projection in parallel, making it possible to do an on-the-fly-reconstruction. The achieved results demonstrate that a complete FDK reconstruction is computed with the CBEA in less than seven seconds for a standard clinical scenario. Given the fact that scan times are usually much higher, we conclude that reconstruction is finished 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.

Keywords: Reconstruction, Computed Tomography, FDK Method, Cone-Beam CT, Back-Projection, Filtering, Cell Processor, Cell Broadband Engine

1. INTRODUCTION

The FDK method[1] is used in most cone-beam CT scanners as the standard reconstruction approach. The computational complexity of this algorithm is still too demanding for today's high performance workstations and, as a consequence, reconstruction time is in the range of minutes and thus not acceptable for many medical work flows. For example the physician is forced to interrupt patient treatment during surgery for several minutes until the reconstruction task completes. Industrial solutions face the performance challenge using dedicated special-purpose reconstruction platforms using digital signal processors (DSPs) and field programmable gate arrays (FPGAs). The most apparent downside of such solutions is the loss of flexibility and their time-consuming implementation, which can lead to long innovation cycles.

The brand-new CBEA[2,3] introduced by IBM, Toshiba, and Sony is a general-purpose processor consisting of a Power Processor Element (PPE) together with eight Synergistic Processing Elements (SPEs) offering a theoretical performance of 204.8 Gflops*(3.2 GHz, 8 SPEs, 4 floating point multiply and add per clock cycle) on a single chip. The processor is still completely programmable using high level languages such as C and C++. The major challenge of porting an algorithm to the CBEA is to exploit the parallelism that it exhibits. The PPE, which is compliant with the 64-bit Power architecture, is dedicated to host a Linux operating system and manages the SPEs as resources for computationally intensive tasks. The SPEs support 128 bit vector instructions

Further author information: (Send correspondence to Holger Scherl)
Holger Scherl: E-mail: scherl@informatik.uni-erlangen.de, Telephone: +49 9131 85 28977

*1 Gflops = 1 Giga floating point operations per second to execute the same operations simultaneously on multiple data elements (SIMD).[4] In case of single precision floating point values (4 bytes each) four operations can be executed at the same time. The SPEs are highly optimized for running compute-intensive code. They have only a small memory each (local store, 256 KB) and are connected to each other and to the main memory via a fast bus system, the Element Interconnect Bus (EIB). The data transfer between SPEs and main memory is not done automatically as is the case for conventional processor architectures, but is under complete control of the programmer, who can thus optimize data flow without any side effects of cache replacement strategies.

In this work, we evaluate the performance of state-of-the-art reconstruction approaches on the Cell processor. We consider the parallelized implementation and optimization of the FDK method and introduce a novel concept for the on-the-fly reconstruction while projection data are acquired.

In the following section we will compare our results to other approaches towards accelerating CT-image reconstruction. Then, we will briefly review FDK reconstruction theory. In section 4 we will describe our CBEA implementation approach in detail, and in section 5 we will report about the achieved results. Finally, we will conclude our work.

2. RELATED WORK

Published results using PC-based implementations still need several minutes for the reconstruction at high spatial resolution.[5–7] Therefore, many specialized hardware platforms have been designed in the past to reconstruct volumes from cone-beam projections, ranging from dedicated hardware solutions like FPGAs[8, 9] and DSPs to clusters of workstations. Even graphics cards for the gaming market are used to perform the back-projection since they can compute the required perspective transformations very efficiently and are available at low prices.[10, 11] However, our CBEA implementation outperforms the mentioned processor-based solutions by one order of magnitude and solutions based on dedicated hardware suffer from rather long development cycles and are often inflexible regarding new requirements.

The Cell processor is a promising alternative to these approaches since it offers an immense computing power due to its high level of parallelism, its high clock rates and its enormous internal communication bandwidth, but can still be programmed using high level programming languages.

A flat-panel cone-beam back-projection was published using one of the two Cell processors of a dual Cell Blade.[7] 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 approach avoids image rectification as suggested[7] that leads to the elimination of the homogeneous division[12] but introduces an additional low-pass operation on the projection. The authors[7] claim that, potentially, on-the-fly-reconstruction could be achieved with the CBEA. In this paper we demonstrate on-the-fly-reconstruction for the complete FDK method (filtering and back-projection) using only one Cell processor in nearest neighbor interpolation mode. When using two Cell processors simultaneously we also achieved on-the-fly-reconstruction in bilinear interpolation mode.

To our knowledge, there is no other group that can handle the challenge of on-the-fly-reconstruction in cone-beam CT. In the next sections we give a detailed overview of how we accomplished on-the-fly-reconstruction capability with our implementation on the CBEA.

3. METHOD

In 1984, Feldkamp, Davis, and Kress[1] published an algorithm for circular cone-beam tomography, which is still widely used in state-of-the-art cone-beam scanning devices; e.g., C-arm CT. It describes an analytical reconstruction method resulting in a filtered back-projection scheme. It can be understood as an extension of exact 2D reconstruction algorithms for fan-beam projections[13] to the three-dimensional case by properly adapting the weighting factors.

The FDK method can be divided into three steps: generate weighted projection data (cosine weighting), ramp filter the projection images row-wise, and back-project the filtered projection data into the volume.[14] For

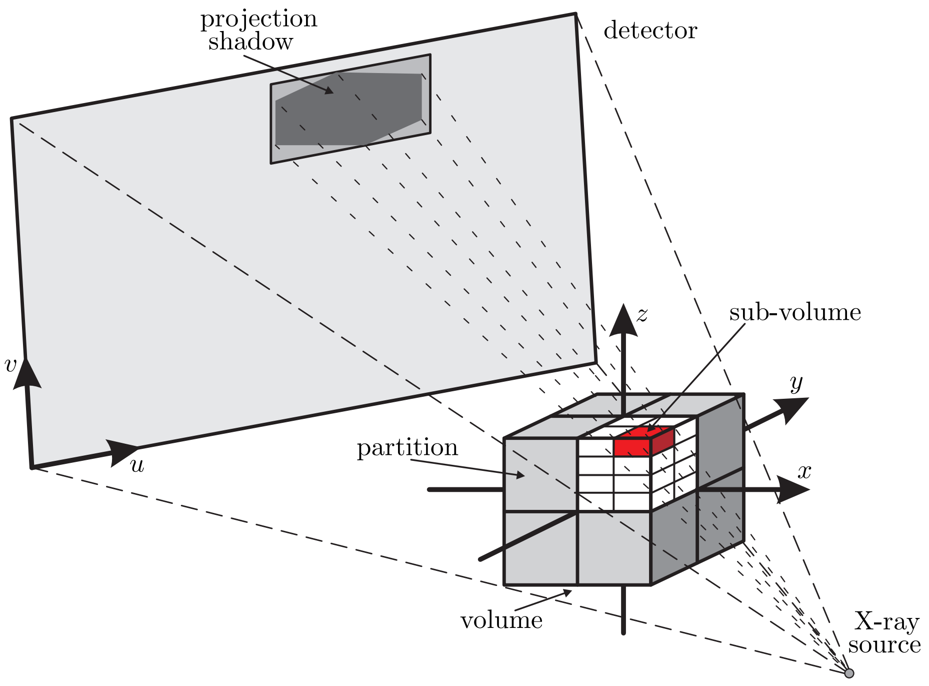

Figure 1. Perspective geometry of the C-arm device (the v-axis and z-axis are not necessarily parallel) together with the parallelization strategy of our back-projection implementation on the Cell processor (partitions are sent as tasks to the SPEs, which are further subdivided in sub-volumes).

short-scan reconstructions an additional weighting (sinogram weighting) has to be applied to account for data redundancy; e.g., Parker Weighting.

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 calculated only once when the C-arm device is installed. As we use a voxel-based back-projection approach it is required to calculate a matrix vector product for each voxel and projection. However, for neighboring voxels it is sufficient to increment the homogeneous coordinates with the appropriate column of P_i.[5] The homogeneous division is still required for each voxel. It cannot be neglected for a line parallel to the z-axis of the voxel coordinate system since, in general, the projection planes are slightly tilted with respect to this axis (see Figure 1 for an illustration of the perspective geometry of a C-arm device). We intentionally avoided to use detector re-binning techniques that align the virtual detector to one of the volume axis since this technique influences the achieved image quality and requires additional computations for the re-binning step.[12]

4. IMPLEMENTATION

The basic processing chain of the FDK algorithm can easily be mapped to a pipeline architecture consisting of dedicated stages. One stage is responsible for loading the projections from the hard disk or over the network. As soon as a projection is available it can be processed by the subsequent stages, which do the weighting, the filtering, and the back-projection. Our software framework extremely simplifies the implementation of such a pipeline. All stages are executed in parallel enabling on-the-fly reconstructions in real-time. In order to utilize the processing elements of the Cell processor, there is integrated support to dispatch the associated parallel processing of a stage to a configurable number of SPEs or PPEs. The PPE acts as the dispatcher which divides the processing of the considered stage into smaller tasks and assigns them to the available processing units. This concept is flexible with respect to the number of SPEs that are available for the processing while others might be busy with other tasks like filtering or preprocessing data.

To minimize the control overhead we assign rather large tasks to the processing elements that further have to be divided into smaller tasks by the processing elements themselves. We take special care to hide any communication latencies via double buffering techniques during the dispatching and computation process. The only downside of our approach is that the mapping of the SPEs onto the stages is currently done statically. This means that we have to decide which SPEs shall be used for the filtering or back-projection stage before program execution. Our approach can be extended to reduce the number of utilized SPEs for each stage dynamically, if higher priority tasks are waiting to be dispatched in other pipeline stages.

An efficient implementation on the CBEA further requires to choose a proper parallelization strategy for each algorithm that can deal with the limited local store size.

Filtering

During the filtering we assign several projection rows to an SPE at the same time. The SPE can process simultaneously two rows by loading them into its local store, applying the sinogram weights and performing the FFT-based convolution after adding the required zero-padding. As we are dealing with real-valued input only we can convolve two image rows simultaneously via computing the complex 1D FFT followed by the multiplication of the DFT of the filter kernel and the computation of the IFFT of the respective product. Communication latencies are effectively hidden via double buffering techniques.

Back-projection

There are basically two options for the parallelization of our voxel-based back-projection approach: projection parallelism and voxel parallelism. Our goal was to create independent tasks that can be flexibly assigned to idle SPEs. We thus decided for voxel parallelism that partitions the volume to be reconstructed and assigns different sub-volumes to the available SPEs. In contrast to projection parallelism there is no need for an additional synchronization step that combines the results of the different SPEs in the end.

Two critical resources have to be considered when creating tasks for an SPE: the small amount of local memory and the limited communication bandwidth between main memory and the SPEs. One basic back-projection task takes a small sub-volume and the associated projection data as input. The updated sub-volume data is written back to main memory after computation. The projection data which a sub-volume depends on is given by the convex hull of the sub-volume corner points projected onto the image plane (see Figure 1). This region will be referred to as the 'projection shadow' of the sub-volume. While the memory requirements for sub-volume data are determined by its dimensions only, the size of the projection shadow depends on a variety of parameters of the acquisition geometry, the discretization (voxel and pixel size), and the shape of the sub-volume, together with the position of the sub-volume within the volume of interest (VOI).

The sub-volume shape was optimized by simulating the maximum projection shadow size and the overall amount of projection data that has to be transferred for a given acquisition geometry and discretization. It turned out that a suitable sub-volume shape is large in x- and y-direction, but small in z-direction. This can be intuitively explained by the fact that the main direction of projection rays is roughly parallel to the x-y-plane and therefore each pixel of the projection shadow is required more often in the computation for that sub-volume. We use a sub-volume size of $32 \times 32 \times 8$ voxels, which is a good trade-off between memory requirements and overall amount of data transfer.

To decrease the communication of volume data between main memory and the SPEs, we queue a few projection images for simultaneous processing in order to reduce the total amount of data transfers. To further speed up access in main memory, we store the sub-volumes sequentially. So the complete volume is stored sub-volume by sub-volume, instead of line by line, plane by plane in main memory. In order to mitigate TLB-thrashing we use huge pages of 16 MB size. By applying the double buffering technique, where the computation uses one data buffer while the other one is transferred using a DMA command, we are able to hide data transfer times completely for the back-projection of high-resolution volumes.

For an efficient implementation of an algorithm on the SPEs one has to exploit their SIMD capabilities and enable a good instruction scheduling to the two pipelines of each SPE. When looking at the back-projection for a single voxel from one projection image, one can identify the following three steps:

Interpolation mode	nearest neighbor		bilinear unoptimized		bilinear optimized	
SPE execution pipeline	even	odd	even	odd	even	odd
Address computation	14	1	17	1	15	1
Projection access	0	14	10	56	10	30
Voxel increment	3	2	3	2	3	2
Total	17	17	30	59	28	33

Table 1. Numbers of instructions required for the back-projection of a vector of voxels from one projection image.

- Compute the projection coordinates and the address of the associated projection pixel.
- Read the value of that pixel using nearest neighbor or bilinear interpolation.
- Increment the voxel value appropriately.

The first and the last step can be vectorized by performing the computations for multiple voxels in parallel. We choose to use vector instructions for neighboring voxels in x-direction, because each of the sub-volumes is stored in that order. The second step cannot be vectorized, because the required projection values will usually not be located in consecutive and aligned memory as required by a vector operation. Each SPE can issue simultaneously in each cycle (vector) instructions into two different pipelines. The even pipeline performs fixed and floating point arithmetic while the odd pipeline executes only load, store, and byte permutation operations. By a proper scheduling of instructions, an optimal cycles per instruction (CPI) ratio of 0.5 can be achieved. Table 1 shows the numbers of instructions required for our algorithm with different interpolation modes on the two pipelines of the SPEs. While the address computation and voxel increment mainly require arithmetic instructions that are executed on the even pipeline, the projection data access is executed on the odd pipeline. We applied loop-unrolling techniques to the iteration over the voxels of a sub-volume in order to leverage efficient instruction scheduling and achieved code with a CPI ratio of 0.57 for the case of nearest neighbor interpolation.

Note the high number of instructions required for the data access. This is due to the fact that the SPEs can not access basic data elements randomly in a vectorized way. Up to four instructions are required to load a single value: rotate the address into the preferred slot of a vector register, load the appropriate vector from memory, rotate the required element into the preferred slot, and finally shuffle it into the destination slot of the destination vector.[15] This is especially a problem for bilinear interpolation since, in this case, four projection pixels have to be accessed for each voxel update. This results in poor performance, if bilinear interpolation is implemented in a straightforward manner. We decreased the number of instructions required for the memory access during the back-projection by adapting the data layout of the loaded projection shadow before performing the actual back-projection. Therefore, we duplicated for each projection pixel the neighboring column pixel value into the same vector. This allows us to access two values with just one vector load instruction and therefore requires only two memory accesses per voxel. The data layout adaption can be performed at low computational cost because it can be vectorized efficiently. The drawback of this method is of course that it requires more memory to store the projection shadow on the SPEs.

5. RESULTS

The filtering and back-projection code was executed on a Blade server board based on the Cell architecture. The board comprises two Cell processors running at 3.2 GHz each as well as 1 GB of main memory split across the two chips. The measurements were done using two data sets. Dataset (a) consists of 414 projections of 1024×1024 pixels each. Because of the fact that, nowadays, larger data sets are also used for the reconstruction we therefore included dataset (b) with 543 projections of 1240×960 pixels each.

In order to assess the performance of our implementation we used the `gettimeofday` function on the PPE. This ensures that all overhead during program execution (e.g., starting the SPE threads) are included in the

Number of SPEs	1	2	3	8	16
Dataset (a), convolution length 2048					
Time [s]	5.84	2.97	1.99	0.82	0.49
Speed-up	1.00	1.97	2.93	7.09	11.80
pps	70.93	139.44	207.95	503.03	836.80
Dataset (b), convolution length 4096					
Time [s]	14.64	7.35	4.91	1.89	1.02
Speed-up	1.00	1.99	2.98	7.75	14.42
pps	37.09	73.84	110.62	287.50	534.96

Table 2. Performance results of FFT-based filtering for the two datasets (convolution length of dataset (a) is 2048 and of dataset (b) is 4096 due to zero-padding).

measurements. In contrast, runtime measurements solely relying on the SPE decrementer (performance counter on SPE side) do only include the SPE program runtime, and a representative of all distinct SPE measurements has to be chosen. We perceived that measurements relying on the SPE decrementer lead to slightly reduced runtime (approximately 0.1 to 0.2 seconds below the runtime measured with `gettimeofday`).

During our measurements we removed any outliers by taking only the best runtime out of five measurements, although runtime deviations of our experiments were only around 0.01 seconds. Care was taken to exclude any influence of other significant PPE or SPE workload. After the correctness of the implementation was verified, we performed the measurements without doing the I/O transfers for loading the projection images from the hard disk or over the network. This was necessary, in order to achieve runtime measurements that were not affected by I/O bandwidth limitations in our current Cell Blade evaluation system.

In a first step, we measured the performance of the filtering code and the back-projection code separately. Finally, we validated the performance of the overall pipeline execution (simultaneous, parallel execution of filtering and back-projection in a pipeline) for various partitioning configurations.

Filtering

The FFT-based filtering required different convolution lengths for the two datasets. Dataset (a) was filtered with convolution length of 2048 and dataset (b) was filtered with a convolution length of 4096 including zero-padding. Table 2 shows the corresponding results for the filter execution using 1, 2, 3, 8 and 16 SPEs. The speed-up factor relative to the execution with only one SPE is also given, together with the number of projections that were processed in one second (pps). Using all 16 SPEs filtering can be done for dataset (a) in 0.49 seconds and for dataset (b) in 1.02 seconds. The FFT computations accounted for 86.71% of the total processing time for dataset (a) and for 90.78% of the total processing time for dataset (b). Sinogram weighting took 3.53% and 2.07% of the processing time for dataset (a) and dataset (b), respectively. Data transfer time was negligible for dataset (b), but consumed 1.11% of processing time for dataset (a) and even increased approximately to 5% if all 16 SPEs are used for the computation. This is the reason why speed-up factors do not scale up linearly, especially not for the smaller dataset.

Back-Projection

We back-projected the cone-beam projections of the two datasets under consideration into a volume consisting of $512 \times 512 \times 512$ voxels. On condition that all voxels were inside the field of view, we chose the largest possible isotropic voxel size. When reconstructing dataset (a) the voxel size was 0.26^3 mm^3, and for dataset (b) the voxel size was 0.31^3 mm^3. In Table 3 we show the achieved results when executing only the back-projection using up to 16 SPEs of our dual Cell Blade for both datasets. The results have been measured in nearest neighbor and in bilinear interpolation mode. Again, we give the speed-up factors relative to the execution with only one SPE and the number of projections that can be back-projected per second (pps). For convenience, we also calculated the number of 512×512 volume slices, that can be reconstructed in one second (frames per second, fps).

Number of SPEs	1	5	6	7	**8**	13	14	15	**16**
Dataset (a), nearest neighbor interpolation									
Time [s]	93.13	18.80	15.71	13.50	**11.85**	7.42	6.89	6.47	**6.07**
Speed-up	1.00	4.95	5.93	6.90	**7.86**	12.56	13.51	14.40	**15.33**
pps	4.45	22.02	26.35	30.67	**34.94**	55.83	60.05	63.99	**68.16**
fps	5.50	27.24	32.58	37.94	**43.21**	69.04	74.26	79.14	**84.29**
Dataset (b), nearest neighbor interpolation									
Time [s]	122.34	24.60	20.53	17.64	**15.44**	9.62	8.97	8.39	**7.89**
Speed-up	1.00	4.97	5.96	6.94	**7.92**	12.71	13.65	14.58	**15.51**
pps	4.44	22.07	26.45	30.79	**35.17**	56.43	60.57	64.73	**68.85**
fps	4.18	20.81	24.94	29.03	**33.16**	53.21	57.11	61.03	**64.92**
Dataset (a), bilinear interpolation									
Time [s]	166.50	33.43	27.91	23.96	**20.99**	13.06	12.12	11.38	**10.68**
Speed-up	1.00	4.98	5.97	6.95	**7.93**	12.75	13.74	14.63	**15.60**
pps	2.49	12.38	14.83	17.28	**19.73**	31.70	34.16	36.38	**38.78**
fps	3.08	15.32	18.34	21.37	**24.40**	39.20	42.25	45.00	**47.96**
Dataset (b), bilinear interpolation									
Time [s]	220.37	44.22	36.88	31.65	**27.72**	17.18	16.02	14.95	**14.02**
Speed-up	1.00	4.98	5.97	6.96	**7.95**	12.83	13.76	14.74	**15.71**
pps	2.46	12.28	14.72	17.16	**19.59**	31.60	33.90	36.32	**38.72**
fps	2.32	11.58	13.88	16.18	**18.47**	29.80	31.97	34.24	**36.51**

Table 3. Performance results of back-projection for the two datasets (nearest neighbor and bilinear interpolation mode).

In nearest neighbor interpolation mode more than 30 projections can be reconstructed per second using only seven SPEs, which is sufficient for on-the-fly-reconstruction that is synchronized to the acquisition. Real-time imaging capability is also achieved in bilinear interpolation mode as soon as more than 12 SPEs are used for the back-projection.

The speed-up factor scales almost linearly, thus indicating that our back-projection implementation was not affected by memory bandwidth limitation. We could not observe communication latencies when profiling our code by instrumenting it with the SPE decrementer (performance counter on SPE side) either.

Right now, we considered only reconstruction volumes that are contained completely inside the field of view. When increasing the voxel size, parts of the cubic voxel volume will be outside the cylindrical field of view (FOV), and hence no meaningful values can be reconstructed in these regions any more. Our optimized implementation takes advantage of this by ignoring any voxels that lie outside the FOV.

In Figure 2 we show the achieved performance of FOV optimized reconstructions using dataset (a) for volumes with an increased voxel size, such that only 78.26%, 85.07%, and 92.56% can be reconstructed, respectively. The achieved frames per second using different numbers of SPEs in nearest neighbor and bilinear interpolation mode are given. The volume size in z-direction had to be adapted such that all slices are still in the field of view. It can be seen that proportional to the reduced number of necessary computations higher frame rates were achieved. Because sub-volumes at the border of the cylindrical FOV still include voxels lying outside of it, the overhead of our implementation increases, as theoretically expected, up to 5% for reconstructing 78.26% of the volume.

For the nearest neighbor interpolation mode the performance degrades slightly due to bandwidth limitation, when reconstructing only 85.07% or 78.26% (maximum speed-up factors were only 14.57 and 14.48, respectively). That's because of the fact that for sub-volumes with increased voxel sizes the corresponding projection shadows will increase significantly. However, in bilinear interpolation mode this increased data transfer overhead was still hidden by the applied double buffering approach. Frame rates increased from 84.29 fps to 95.66 fps (nearest neighbor interpolation mode) and from 47.96 fps to 55.72 fps (bilinear interpolation mode).

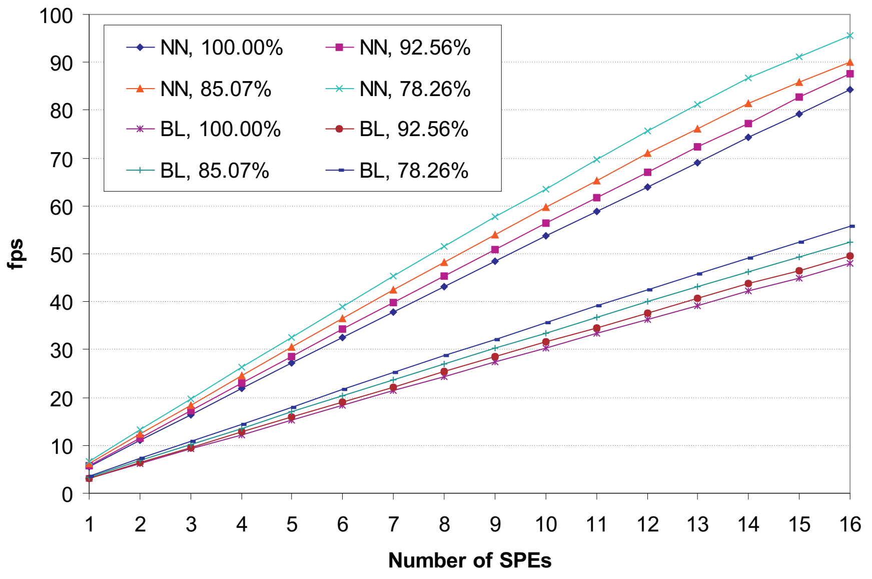

Figure 2. Achieved frames per second (fps) for reconstructions using dataset (a) with different voxel sizes. The percentage of the voxel volume that lies inside the field of view is given for each measurement. Frame rates are given for the two interpolation modes: nearest neighbor (NN) and bilinear (BL) interpolation.

Overall Pipelined Execution

In the next experiment we executed the filtering and the back-projection in parallel in a pipeline. The number of SPEs for filtering and back-projection has to be chosen statically before execution. We examined the overall performance with both nearest neighbor and bilinear interpolation mode. Table 4 shows the corresponding results for various configurations. When only 8 SPEs of our dual Cell Blade are used, it was sufficient to execute the filtering with one SPE for maximum performance for both interpolation modes. The measured overall runtime reflects approximately the measured execution time of the back-projection with seven SPEs. Thus, filtering execution could be fully hidden behind the back-projection. In case of nearest neighbor interpolation more than 30 projections are processed per second, which still leverages an on-the-fly reconstruction using only one Cell processor. If one wants to achieve on-the-fly reconstruction also in bilinear interpolation mode at least two Cell processors have to be used (see Table 3). The achieved performance numbers of the overall pipelined execution validate that on-the-fly reconstruction is possible even in bilinear interpolation mode for all tested configurations of filtering and back-projection SPEs. However, maximum performance was achieved when two SPEs are used in case of dataset (b).

6. CONCLUSIONS

We showed a parallelized and highly optimized implementation of the FDK method on the novel Cell processor. The achieved results demonstrate that a performance increase of an order of magnitude and more is achievable compared to recent high performance general purpose computing platforms. With our dual Cell Blade we can compute a complete FDK short-scan reconstruction (including weighting, filtering and back-projection) in 6.97 seconds (nearest neighbor interpolation) or 11.44 seconds (bilinear interpolation).

This leverages cone-beam CT reconstructions on-the-fly for both interpolation modes, which means that we can hide all required computations behind the scan-time of the used device. As of this writing, we know of no other group who can present the reconstructed volume to the physician immediately after the last projection image has been acquired from the scanner. Concerning real-time imaging, the achieved performance of the FDK method shows that there is still the possibility for computing additional preprocessing tasks such as beam hardening, truncation or scatter correction on the dual Cell Blade.

	Number of SPEs (filtering/back-projection)					
	using one Cell processor			using two Cell processors		
	1/7	2/6	3/5	1/15	2/14	3/13
Dataset (a), nearest neighbor interpolation						
Time [s]	**13.60**	15.75	18.83	7.39	**6.97**	7.45
pps	**30.44**	26.29	21.98	56.06	**59.43**	55.55
fps	**37.64**	32.52	27.18	69.33	**73.50**	68.70
Dataset (b), nearest neighbor interpolation						
Time [s]	**17.87**	20.66	24.69	14.78	**9.07**	9.67
pps	**30.39**	26.28	21.99	36.74	**59.90**	56.16
fps	**28.66**	24.78	20.74	34.64	**56.48**	52.95
Dataset (a), bilinear interpolation						
Time [s]	**24.04**	27.95	33.49	**11.44**	12.19	13.10
pps	**17.22**	14.81	12.36	**36.18**	33.95	31.61
fps	**21.30**	18.32	15.29	**44.74**	41.99	39.09
Dataset (b), bilinear interpolation						
Time [s]	**31.87**	36.99	44.28	16.21	**16.05**	17.22
pps	**17.04**	14.68	12.26	33.49	**33.84**	31.53
fps	**16.06**	13.84	11.56	31.58	**31.91**	29.73

Table 4. Overall pipelined execution of filtering and back-projection for the two datasets (nearest neighbor and bilinear interpolation), bold numbers refer to optimum configurations.

We conclude that especially in flat-panel cone-beam CT (e.g., C-arm devices), the CBEA represents a very promising computing architecture and, due to its possibility to be programmed in a high level programming language, will help to reduce the innovation cycles of the reconstruction system in commercial scanning devices, significantly.

REFERENCES

1. L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *J. Opt. Soc. Amer.* **A1**(6), pp. 612–619, 1984.
2. IBM, *Cell Broadband Engine Programming Handbook*, 1.0 ed., 2006.
3. D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," in *IEEE Solid-State Circuits Conference*, pp. 184–185, (San Francisco), 2005.
4. H.-J. Oh, S. Mueller, C. Jacobi, K. Tran, S. Cottier, B. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. Dhong, "A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor," *IEEE Journal of Solid-State Circuits* **41**(4), pp. 759–771, 2006.
5. 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), pp. 391–403, 2000.
6. R. Yu, R. Ning, and B. Chen, "High-speed cone-beam reconstruction on PC," in *Proc. SPIE Medical Imaging 2001: Image Processing*, M. Sonka and K. Hanson, eds., **4322**, pp. 964–973, (San Diego), 2001.
7. 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.
8. M. Trepanier and I. Goddard, "Adjunct processors in embedded medical imaging systems," in *Proc. SPIE Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display*, S. Mun, ed., **4681**, pp. 416–424, 2002.

9. I. Goddard and M. Trepanier, “High-speed cone-beam reconstruction: an embedded systems approach,” in *Proc. SPIE Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display*, S. Mun, ed., pp. 483–491, 2002.
10. F. Xu and K. Mueller, “Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware,” *IEEE Transactions on Nuclear Science* **52**(3), pp. 654–663, 2005.
11. K. Mueller and F. Xu, “Practical considerations for GPU-accelerated CT,” in *IEEE International Symposium on Biomedical Imaging*, 2006. SU-AM-OS3.3.
12. C. Riddell and Y. Trousset, “Rectification for cone-beam projection and backprojection,” *IEEE Transactions on Medical Imaging* **25**(7), pp. 950–962, 2006.
13. A. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, SIAM, 2001.
14. H. Turbell, *Cone-Beam Reconstruction Using Filtered Backprojection.* PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, February 2001. Dissertation No. 672, ISBN 91-7219-919-9.
15. IBM, *Synergistic Processing Unit Instruction Set Architecture*, 1.0 ed., 2005.