On-the-fly-Reconstruction in Exact Cone-Beam CT using the Cell Broadband Engine Architecture

Holger Scherl, Stefan Hoppe, Frank Dennerlein, Günter Lauritsch, Wieland Eckert, Markus Kowarschik, and Joachim Hornegger

***Abstract*—In medical imaging a high image quality is required. Exact cone-beam reconstruction algorithms, e.g. the M-line method, provide excellent image quality without any cone artifacts. Although the M-line approach is still of a filtered back-projection style, it has an increased computational complexity as it requires additional computations for the filtering of the projection images, e.g. derivative computation and filtering along oblique lines in the projection. In order to enable this new reconstruction methodology in standard clinical scenarios we implemented a highly performance optimized version on the Cell Broadband Engine Architecture (CBEA). Our software framework allows to compute the filtering and back-projection in parallel to the data acquisition, making it possible to do an on-the-fly-reconstruction. The achieved results demonstrate that M-line reconstructions with our optimized Cell-based implementation are finished immediately after the last projection image has been acquired by the scanning device.**

***Index Terms*—Reconstruction, Computed Tomography, M-line Method, Cone-Beam CT, Cell Processor, Cell Broadband Engine Architecture**

I. Introduction

The theoretically exact M-line based reconstruction method [1] totally resolves the problem of cone artifacts. In the state-of-the-art FDK [2] the occurring cone artifacts may cover small object details complicating their distinction. The M-line approach could by applied for C-arm CT [3]. Due to its requirement of a complete cone-beam data acquisition, the source trajectory has to be extended to a short-scan circle-plus-arc acquisition. With respect to computation times, however, the improved image quality does not come for free. In addition to the computation of a differentiated projection image filtering is done along oblique lines. Thus, especially the filtering of projections incurs much more computations to be performed by the image reconstruction hardware. In this work, we evaluate the performance of exact reconstruction methods by means of the M-line approach on the Cell processor.

The novel CBEA [4], [5] 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[1](3.2 GHz, 8 SPEs, 4 floating point multiply-and-add per clock cycle) on a single chip. The PPE manages the SPEs as resources for computationally intensive tasks. The SPEs 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 the following we consider the parallelized implementation and optimization of the M-line method and demonstrate an on-the-fly-reconstruction while projection data are acquired. Our implementation supports the case of a non-ideal data acquisition and can thus reconstruct data sets from real C-arm CT scanners. We also compare the achieved results with our optimized CBEA implementation of the FDK method [6].

H. Scherl, S. Hoppe 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.

F. Dennerlein is with the University of Utah, Department of Radiology (UCAIR), 729 Arapeen Drive, Salt Lake City, UT 84108, USA.

G. Lauritsch and W. Eckert are with Siemens Medical Solutions, AX Division, P.O.Box 1266, D-91294 Forchheim, Germany.

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

II. Reconstruction Algorithm

The task in image reconstruction is to recover the density of an object $f(\underline{x})$ under examination provided a set of line integrals

$$g(\lambda, \underline{\theta}) = \int_0^\infty f(\underline{a}(\lambda) + t\underline{\theta})dt, \tag{1}$$

where λ denotes the source trajectory parameter, $\underline{a}(\lambda)$ describes the corresponding source position and $\underline{\theta}$ the direction of the line. If we assume a flat panel detector located at a distance D from the current source position, each detector value at coordinates $(u, v)^T$ refers to a line integral with line direction

$$\underline{\theta}(u, v) = (u\underline{e}_u + v\underline{e}_v - D\underline{e}_w) / \sqrt{u^2 + v^2 + D^2}. \tag{2}$$

Here, the detector coordinates are identified by two unit normal vectors $\underline{e}_u$ and $\underline{e}_v$ and the coordinate origin $(0, 0)^T$ is the orthogonal projection of $\underline{a}(\lambda)$ onto the detector. The vector $\underline{e}_w = \underline{e}_u \times \underline{e}_v$ points from the detector towards the source position. To reconstruct a point $\underline{x}$ inside the support of the object with the M-line approach, the following steps are applied successively. The computations are performed for each trajectory segment q, with $\lambda_q^- \leq \lambda \leq \lambda_q^+$, and then all contributions are accumulated according to Formula C (32) of [1]. Because other exact cone-beam reconstruction algorithms, as e.g. [7], can be implemented in a similar way, M-line specific steps are marked as such in the following. As

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

can be seen only the rebinning steps had to be modified in order to account for other filtering directions.

Step 1 - Filtering: Each projection $g(\lambda, \theta(u,v))$ is modified into a filtered projection $g^F(\lambda, u, v)$ according to the following steps:

F1 - Derivative: Compute the derivative of $g(\lambda, \underline{\theta}(u,v))$ with respect to λ along a constant viewing direction $\underline{\theta}$

$$g_1(\lambda, u, v) = \frac{\partial}{\partial \lambda} g(\lambda, \underline{\theta}(u,v)) \Big|_{\theta = \text{fix}}, \tag{3}$$

with $\underline{\theta}$ defined in (2). This derivative is computed similar to formula (87) of [8].

F2 - Cosine Weighting: Weight the data according to

$$g_2(\lambda, u, v) = \frac{D}{\sqrt{u^2 + v^2 + D^2}} g_1(\lambda, u, v). \tag{4}$$

F3 - Forward Rebinning (M-line specific): Perform a forward rebinning from detector coordinates (u, v) to filter line coordinates (u, s), where s identifies the slope of the filter line, according to

$$g_3(\lambda, u, s) = g_2(\lambda, u, v(u,s)), \tag{5}$$

where

$$v(u, s) = s(u - u_M) + v_M. \tag{6}$$

Here, all filter lines converge to the common point $(u_M, v_M)^T$. Consequently, each filter line is uniquely identified by its slope and the mapping is reversible.

F4 - Hilbert Filtering: Perform a Hilbert transform with respect to u by computing

$$g_4(\lambda, u, s) = \int_{-\infty}^{+\infty} \frac{1}{\pi(u - u')} g_3(\lambda, u', s) du'. \tag{7}$$

F5 - Backward Rebinning (M-line specific): In order to achieve a computational more efficient back-projection, we avoid the direct back-projection from the rebinned grid. Instead we perform a backward rebinning from filter line coordinates (u, s) to detector coordinates (u, v), according to

$$g_5(\lambda, u, v) = g_4(\lambda, u, s(u,v)), \tag{8}$$

where

$$s(u, v) = \frac{v - v_M}{u - u_M}. \tag{9}$$

F6 - π-Weighting: Perform π weighting according to

$$g^F(\lambda, u, v) = m(\lambda, u, v) g_5(\lambda, u, v). \tag{10}$$

The function $m(\lambda, u, v)$ takes only values of one and zero and should be understood as a 2-D weighting mask that accounts for a correct handling of the back-projection segment for each point $\underline{x}$. It can be precomputed once after C-arm geometry calibration as shown in [3] (Section VI).

Step 2 - Back-projection: Back-project the filtered projection $g^F(\lambda, u, v)$ into the image space to obtain f at each point $\underline{x} = (x, y, z)$ according to

$$f(\underline{x}) = -\frac{1}{2\pi} \int_{\lambda_q^-}^{\lambda_q^+} \frac{1}{(\underline{x} - \underline{a}(\lambda))\underline{e}_w} g^F(\lambda, u(\lambda, \underline{x}), v(\lambda, \underline{x})) d\lambda, \tag{11}$$

where u and v are the detector coordinates of $\underline{x}$ given by

$$u(\lambda, \underline{x}) = -D \frac{(\underline{x} - \underline{a}(\lambda))\underline{e}_u}{(\underline{x} - \underline{a}(\lambda))\underline{e}_w} \tag{12}$$

$$v(\lambda, \underline{x}) = -D \frac{(\underline{x} - \underline{a}(\lambda))\underline{e}_v}{(\underline{x} - \underline{a}(\lambda))\underline{e}_w}. \tag{13}$$

III. Implementation

We implemented the basic processing chain of the M-line algorithm as a pipeline consisting of dedicated stages. One pipeline 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 pipeline stages. Our software framework extremely simplifies the implementation of such a pipeline. All pipeline stages are executed in parallel enabling on-the-fly-reconstructions in real-time. The processing elements of the Cell processor are utilized by dispatching the associated parallel processing of a pipeline stage to a configurable number of SPEs. The PPE acts as the dispatcher which divides the processing of the considered pipeline stage into smaller tasks and assigns them to the available processing units. 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 available SPEs onto the pipeline stages is currently done statically. This means that we have to decide how many SPEs shall be used in each pipeline stage before program execution. Assigning each filtering step (F1 to F6) a separate pipeline stage and thus at least one SPE would result in not fully utilized SPEs, which is a waste of computation resources. Technically, we compiled all filtering steps in one pipeline stage and one associated SPE program in order to circumvent this problem. The dispatching PPE identifies each filtering task via a special tag such that a filtering SPE can easily decide which processing task should be executed. Fortunately, together with necessary data buffers the complete SPE program fits into the local store, when temporary data buffers were shared among the different filtering task implementations. The PPE-side dispatching facility of the filtering pipeline stage takes care of synchronization and load-balancing between the individual filtering steps. Because of local store size restrictions, the filtering and back-projection tasks, however, had to be separated into two different pipeline stages and thus also two different SPE programs.

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

F1 and F2 - Derivative and Cosine Weighting: The derivative computation is implemented in a row-based manner. Several rows of the resulting derived projection are assigned to an SPE at the same time. The SPE itself transfers the required rows of each involved projection (our current approximation of the derivative requires the considered projection together with the previous and next projection) to its local store, before performing the actual computations. Because the required computations for the derivative and the cosine weighting share a common factor we could easily combine the involved computations in order to achieve more efficiency.

F3 - Forward Rebinning: The rebinning computations could not be implemented in a line-based manner due to the limited size of the local store. An efficient parallelization strategy must further take into account that optimal sizes for memory transfers on the Cell processor are multiples of 128 bytes (32 single precision floating point values). We therefore decided to partition the rebinned image into blocks of 32×32 values. For each block the corresponding maximum shadow in the projection image is obtained by applying the rebinning equation to the four border values of a block (see figure 1). Then we clip the resulting shadow with the detector boundary and transfer the associated data from main memory to the local store. After that the rebinning computations can be applied on the chosen partition and the resulting values can be transferred back to main memory. In order to avoid communication overhead we let the dispatching PPE assign several blocks to one SPE at the same time. The results are saved to a temporary buffer allocated in main memory.

F4 - Hilbert Filtering: Again, we assign several projection rows to a filtering SPE at the same time. The SPE processes simultaneously two rows by loading them into its local store and performing the convolution based on the fast Fourier transform (FFT) 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 discrete Fourier transform (DFT) of the filter kernel and the computation of the IFFT of the respective product. The DFT of the spatial Hilbert kernel does only have imaginary parts, which simplifies the complex multiplication.

F5 - Backward Rebinning: The backward rebinning step is implemented using the parallelization strategy of the forward rebinning step, in reversed order.

F6 - π-Weighting: For each projection the associated π-weighting mask has to be initialized, e.g. by loading it from the hard disk. In our current implementation we assumed that we have enough bandwidth available for loading the mask images. While most of the mask values have the same value (one or zero), a very simple compression scheme, e.g. based on run-length encoding would easily remove any possible bandwidth limitations. The π-weighting step itself is easily parallelized because it is nothing else than an element-wise multiplication of two 2D arrays.

Step 2 - Back-projection: Since we moved all required M-line specific operations out of the back-projection loop, it was possible to use the same implementation approach for the back-projection as in our highly optimized Cell-based FDK implementation. For further details we refer the reader to [6].

Processing task	Time [s]	Percentage [%]
Derivative/Cosine Weighting	6.40	21.5
Forward rebinning	6.64	22.3
Hilbert Filtering	10.00	33.7
Backward rebinning	5.88	19.8
π-weighting	0.79	2.7
Total	29.71	100.00

TABLE I
PERFORMANCE RESULTS OF THE COMPLETE FILTERING WITHIN THE M-LINE APPROACH USING ONE SPE.

IV. RESULTS

We evaluated the performance of our implementation 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 execution time of our M-line implementation was measured using a data set consisting of 600 projection images of 1024×1024 pixels each. The number of projection images were 500 on the short-scan circle, 50 on the upper arc segment and also 50 on the lower arc segment. The average number of filter lines per projection image was 1071. To achieve computation times that are not affected of field-of-view (FOV) handling strategies we back-projected the cone-beam projections under consideration into a volume that is completely inside the FOV. Therefore, we used a volume consisting of $512 \times 512 \times 352$ voxels with a voxel size of 0.31^3 mm^3. During our measurements we removed any outliers by taking only the best runtime out of five measurements. Care was taken to exclude any influence of other significant PPE or SPE workload of the system. After the correctness of the implementation was verified, we performed the measurements without doing the I/O transfers for loading the projection and mask 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 of our current Cell Blade evaluation system.

In order to measure the execution time of each filtering step separately, we instrumented our code with SPE decrementer statements (performance counter on SPE side). Table I lists the execution time for filtering all 600 acquired projection images of the complete acquisition using one SPE. We took special care to avoid the influence from any other workload. The Hilbert filtering amounts for more than 33 % of the overall filtering computations. While it is implemented in the same way as the filtering of our Cell-based FDK implementation [6] the computing time of filtering increases by a factor of three in the exact approach. Due to the random memory accesses during forward and backward rebinning, the corresponding computations are the most expensive ones. Usually there are more filter lines than rows in the projection images. Because of this reason the forward rebinning accounts for more processing time in comparison to the backward rebinning,

During the validation of the performance of the overall pipeline execution (simultaneous, parallel execution of filtering and back-projection in a pipeline) 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 measurements. Table II shows

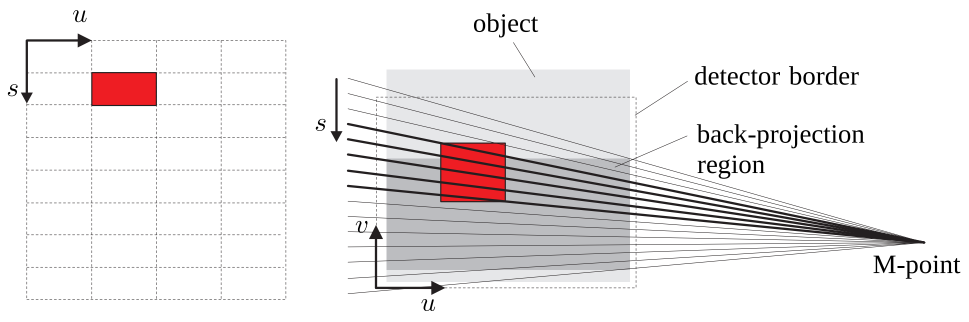

Fig. 1. Principle of forward rebinning. The colored box of the rebinned filter lines (left) correspond to the bold filter lines in the colored box of the projection (right). It is only required to rebin filter lines going through the back-projection region (see F6 for a definition).

	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	4/12
M-line (short-scan circle plus two arc segments)							
Time [s]	33.25	**30.05**	35.24	31.20	17.38	**14.64**	15.61
pps	18.05	**20.00**	17.03	19.23	34.52	**40.98**	38.44
fps	10.59	**11.71**	9.99	11.28	20.25	**24.04**	22.55
FDK (short-scan circle only)							
Time [s]	**19.80**	22.99	27.52	**9.43**	10.07	10.81	
pps	**25.15**	21.66	18.10	**52.81**	49.45	46.07	
fps	**17.78**	15.31	12.79	**37.33**	34.96	32.56	

TABLE II

OVERALL PIPELINED EXECUTION OF THE FILTERING AND BACK-PROJECTION FOR THE M-LINE APPROACH AND THE FDK METHOD.

the achieved results for various configurations of used SPEs for filtering and back-projection, respectively. For comparison purposes we also computed FDK reconstructions with the same SPE configuration using only the projection images from the short-scan circle. We also give the number of projection images that can be processed per second (pps). This number is important because on-the-fly-reconstruction as demonstrated in [6] can only be achieved when the reconstruction system is able to process at least the same number of projections per second than the scanning device can deliver. Recent C-arm devices achieve rates of 30 pps for 1k images. For convenience, we also calculated the number of 512×512 image slices, that can be reconstructed in one second (frames per second, fps) as this number is often used in research for comparison purposes. One can see that, in contrast to the FDK method, the reconstruction speed of the M-line method is limited by the processing time of the SPEs used for the filtering pipeline stage. Using only one Cell processor of our dual Cell Blade two filtering SPEs and using both Cell processors even three filtering SPEs are required in order to achieve optimal performance. Otherwise the execution time is limited by the back-projection performance, which is roughly comparable to the one used in the FDK implementation. Both reconstruction approaches, however, achieve on-the-fly-reconstruction using both Cell processors.

V. CONCLUSIONS

We showed a parallelized and highly optimized implementation of an exact cone-beam reconstruction method on the Cell processor. With our dual Cell Blade we can compute an M-line reconstruction for a standard clinical scenario in 14.64 seconds (40.98 pps) using a short-scan circle plus two arcs acquisition. This leverages high quality cone-beam CT reconstructions on-the-fly, which means that we can hide all required computations behind the scan-time of the used device.

We conclude that in flat-panel cone-beam CT (e.g., C-arm devices), the CBEA is able to give advent to the exact cone-beam reconstruction methods in practical scanning devices.

ACKNOWLEDGMENTS

This work was supported by Siemens AG, Medical Solutions.

REFERENCES

[1] J. Pack and F. Noo, “Cone-beam reconstruction using 1D filtering along the projection of M-lines,” *Inverse Problems*, vol. 21, no. 3, pp. 1105–1120, 2005.

[2] L. A. Feldkamp, L. C. Davis, and J. W. Kress, “Practical cone-beam algorithm,” *J. Opt. Soc. Amer.*, vol. A1, no. 6, pp. 612–619, 1984.

[3] S. Hoppe, F. Dennerlein, G. Lauritsch, J. Hornegger, and F. Noo, “Cone-beam tomography from short-scan circle-plus-arc data measured on a C-arm system,” in *IEEE Nuclear Science Symposium Conference Record*, San Diego, 2006, pp. 2873–2877.

[4] *Cell Broadband Engine Programming Handbook*, 1st ed., IBM, 2006.

[5] 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*, San Francisco, 2005, pp. 184–185.

[6] 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*, J. Hsieh and M. Flynn, Eds., vol. 6510, San Diego, February 2007.

[7] A. Katsevich, “Image reconstruction for the circle-and-arc trajectory,” *Physics in Medicine and Biology*, vol. 50, no. 10, pp. 2249–2265, 2005.

[8] F. Noo and D. Heuscher, “Exact helical reconstruction using native cone-beam geometries,” *Physics in Medicine and Biology*, vol. 48, no. 23, pp. 3787–3818, 2003.