Evaluation of State-of-the-Art Hardware Architectures for Fast Cone-Beam CT Reconstruction

Holger Scherl[a,*], Markus Kowarschik[b], Hannes G. Hofmann[c], Benjamin Keck[c], Joachim Hornegger[c]

[a] *Siemens AG, Healthcare Sector, CV Division, Medical Electronics and Imaging Solutions, Mozartstr. 57, 91052 Erlangen, Germany.*
[b] *Siemens AG, Healthcare Sector, Angiography and Interventional X-Ray Systems, Siemensstr. 1, 91301 Forchheim, Germany.*
[c] *Friedrich-Alexander-University Erlangen-Nuremberg, Department of Computer Science, Pattern Recognition Lab (LME), Martensstr. 3, 91058 Erlangen, Germany.*

Abstract

We present an evaluation of state-of-the-art computer hardware architectures for implementing the FDK method, which solves the 3-D image reconstruction task in cone-beam computed tomography (CT). The computational complexity of the FDK method prohibits its use for many clinical applications unless appropriate hardware acceleration is employed. Today's most powerful hardware architectures for high-performance computing applications are based on standard multi-core processors, off-the-shelf graphics boards, the Cell Broadband Engine Architecture (CBEA), or customized accelerator platforms (e.g., FPGA-based computer components).

For each hardware platform under consideration, we describe a thoroughly optimized implementation of the most time-consuming parts of the FDK algorithm; the filtering step as well as the subsequent back-projection step. We further explain the required code transformations to parallelize the algorithm for the respective target architecture. We compare both the implementation complexity and the resulting performance of all architectures under consid-

*Corresponding author

Email addresses: holger.scherl@siemens.com (Holger Scherl), markus.kowarschik@siemens.com (Markus Kowarschik), hannes.hofmann@informatik.uni-erlangen.de (Hannes G. Hofmann), keck@informatik.uni-erlangen.de (Benjamin Keck), joachim.hornegger@informatik.uni-erlangen.de (Joachim Hornegger)

eration using the same two medical datasets which have been acquired using a standard C-arm device.

Our optimized back-projection implementations achieve at least a speedup of 6.5 (CBEA, two processors), 22.0 (GPU, single board), and 35.8 (FPGA, 9 chips) compared to a standard workstation equipped with a quad-core processor.

Keywords:
Computed tomography, FDK reconstruction, back-projection, filtering, hardware acceleration

---

1. Introduction

The FDK method [1] is used in most of today's cone-beam CT scanners—such as C-arm devices, radiation therapy devices, dental CT devices, and in a modified way also in digital breast tomosynthesis (DBT) devices—as the standard image reconstruction approach. Examples of reconstructed images are given in Appendix A. The typical clinical workflow requires high-speed reconstructions in order to enable high patient throughput or to avoid an interruption of patient treatment during interventional procedures. From the physician's perspective, it is required that the computation of the reconstructed volume from a set of acquired two-dimensional X-ray projections terminates roughly with the end of the scanning period such that no additional time delay is experienced and the volume data set can be analyzed immediately after the scan. In order to fulfill the physician's challenging performance requirements, it is inevitable to utilize fast computing hardware.

The amazing progress in very-large-scale integration (VLSI) design has led to the development of microprocessors consisting of several independent compute cores that can execute multiple application tasks in parallel. The cores belonging to one *central processing unit (CPU)* often share certain levels of the on-chip memory hierarchy (e.g., on-chip caches). These processors are commonly referred to as *multi-core* or even as *many-core* CPUs. CPU manufacturers such as Intel and AMD currently provide up to twelve compute cores on a single chip with forecasts predicting 32 and even more parallel cores per CPU chip.

The *Cell Broadband Engine Architecture (CBEA)*, developed by IBM, Sony, and Toshiba consists of a control core *(Power Processing Element, PPE)* and eight high-performance processing cores *(Synergistic Processing*

Elements, SPEs). The programming methodology of the CBEA, however, poses a major challenge to software developers by demanding careful hand-tuning of programs to get the maximum performance out of this chip.

Standard graphics boards based on powerful *graphics processing units (GPUs)* can serve as another hardware alternative for high-performance computing applications. Modern GPUs in general have evolved into highly parallel, multi-threaded many-core processor architectures accompanied by high-bandwidth graphics memory. The graphics board is a physically separated *device* that operates as a co-processor to the CPU *(host)*. The *Compute Unified Device Architecture (CUDA)* of NVIDIA GPUs provides an easy-to-use computing paradigm that enables programmers to exploit the processing power of GPUs without requiring expertise in computer graphics.

Reconfigurable microchips represent a fourth architecture alternative for accelerating computationally intensive algorithms. *FPGAs (field-programmable gate arrays)* and *CPLDs (complex programmable logic devices)* are the major representatives of reconfigurable circuits [2]. In general, FPGAs are characterized by a larger number of arithmetic units than CPLDs. Several independent vendors offer FPGA-based accelerator cards for use in general-purpose computer environments, along with suitable development kits for software and firmware.

Published results using single- and multi-core CPU-based implementations still need more than several minutes for the reconstruction of volumes with high spatial resolutions of 512^3 or more voxels [3, 4, 5]. Therefore, many specialized hardware platforms were designed in the past to reconstruct volumes from cone-beam projections, ranging from dedicated hardware solutions that employ FPGAs [6, 7, 8] and DSPs to clusters of workstations.

Recently, a flat-panel cone-beam back-projection algorithm was published using one of the two Cell processors of a dual Cell Blade [9]. The results are comparable to the performance of our Cell-based back-projection module. The same implementation approach was used for an OpenGL-based GPU implementation [10, 11].

The most time-consuming instruction of the reconstruction algorithm is the homogeneous divide operation in the innermost loop caused by the perspective projection model. Our approach does not use projection rebinning [9, 10, 11] which, on the one hand, eliminates the divide operation [12] but, on the other hand, introduces an additional implicit low-pass filtering of the projection data.

In order to implement the back-projection on GPUs, OpenGL and shad-

ing languages are also used frequently [13, 14]. Unfortunately, a straightforward comparison of published results is rarely possible in an objective manner since different algorithms, datasets, and acquisition geometries are considered. Furthermore, only few publications address both time-consuming reconstruction tasks (i.e, filtering and back-projection) simultaneously [3, 4].

This article presents a detailed overview of the aforementioned hardware platforms and their application to the 3-D image reconstruction task in cone-beam CT. We will demonstrate the reconstruction speed possible on today's most relevant hardware architectures. In all our benchmarks we used two medical datasets from a standard cone-beam CT scanner.

2. Feldkamp Algorithm

The task in 3-D image reconstruction is to recover the function of X-ray attenuation coefficients $f(\underline{x})$ of an object under examination, provided a set of line integrals

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

Here, the 3-D curve $\underline{a}(\lambda)$ describes the corresponding position of the X-ray source with λ varying over a finite interval of $\mathbb{R}$, and the unit vector $\underline{\theta}$ represents the direction of the respective ray. The Feldkamp algorithm is based on a circular trajectory, see Figure 1. Each point $\underline{a}(\lambda)$ on the source path at rotation angle λ (expressed in radians) of the source-detector assembly is given by the vector

$$\underline{a}(\lambda) = (R\cos\lambda, R\sin\lambda, 0)^T \,,\ \lambda \in [0, 2\pi[\,, \tag{2}$$

where the coordinates on the right-hand side refer to the fixed right-handed world coordinate system (WCS) defined by its origin O and the unit vectors $\underline{e}_x = \underline{e}_x(\lambda)$, $\underline{e}_y = \underline{e}_y(\lambda)$, and $\underline{e}_z = \underline{e}_z(\lambda)$. If we assume a flat-panel detector located at a distance D from the current source position, the detector value at coordinates $(u, v)^T$ refers to an integral $g(\lambda, u, v)$ along a straight line with direction

$$\underline{\theta}(u, v) = \left(u\underline{e}_u + v\underline{e}_v - D\underline{e}_w\right) / \sqrt{u^2 + v^2 + D^2} \,. \tag{3}$$

The detector coordinates are identified by two orthogonal unit vectors $\underline{e}_u$ and $\underline{e}_v$ and the origin $(0, 0)^T$ of the detector coordinate system (DCS) which is determined by the orthogonal projection of $\underline{a}(\lambda)$ onto the detector plane.

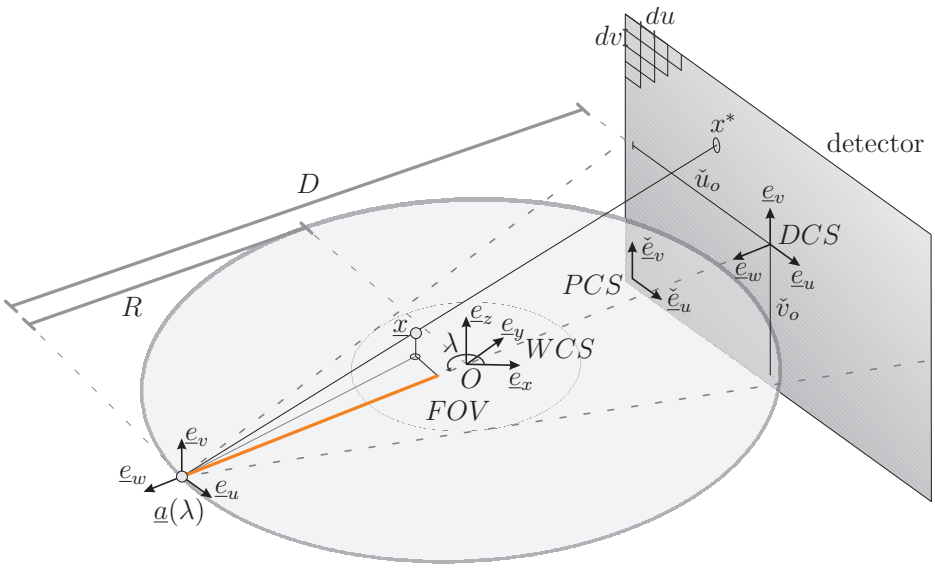


Figure 1: Ideal Feldkamp geometry. The X-ray source rotates along an ideal circle of radius R with center point O in the midplane about the axis of rotation that is defined by the point O and the direction $\underline{e}_z$. The rotation direction is given by the angle λ in counterclockwise direction. D is the orthogonal distance between $\underline{a}(\lambda)$ and the corresponding detector plane. The cylindrical field-of-view (FOV) is indicated by the inner circle. Only points inside the FOV can be reconstructed.

The unit vector $\underline{e}_w = \underline{e}_u \times \underline{e}_v$ points from the origin of the DCS towards the source. Throughout this article we use the notation $\check{\cdot}$ to denote quantities that refer to coordinates of the pixel coordinate system (PCS) or the voxel coordinate system (VCS).

2.1. Algorithmic Steps

In order to approximately reconstruct a value $f(\underline{x})$ at any position $\underline{x}$ within the support of the object density function f using the Feldkamp algorithm, the following algorithmic steps must be applied successively [1]. Note that this description of the algorithmic steps is based on a continuous model of the CT device. In practice, however, projection images are only acquired at a limited number of projection angles. Moreover, a detector consists of discrete sensor elements that are arranged in a matrix structure of rows and columns. Additionally, several preprocessing steps (e.g. intensity and scatter correction) are applied on the projection images before reconstruction (see [15] for more details).

Step 1 – Filtering.. Each projection $g(\lambda, u, v)$ is transformed into a filtered projection $g^F(\lambda, u, v)$ according to the following steps F1 and F2:

F1 – Cosine Weighting.. Weight the projection data according to

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

where the factor $D/\sqrt{u^2 + v^2 + D^2}$ represents the cosine of the angle between the principal ray of the cone-beam hitting the detector in the origin of the DCS and the ray hitting the detector at position $(u, v)^T$, cf. Figure 1.

F2 – Ramp Filtering.. Ramp-filter the projection images row-wise (i.e., with respect to u) by computing

$$g^F(\lambda, u, v) = g_1(\lambda, u, v) * h_{ramp}(u)\,, \tag{5}$$

where h_{ramp} denotes the ideal ramp filter (see Figure 2) and $*$ represents the convolution operator. This step corresponds to a 1-D convolution along lines on the detector that are parallel to $\underline{e}_u(\lambda)$.

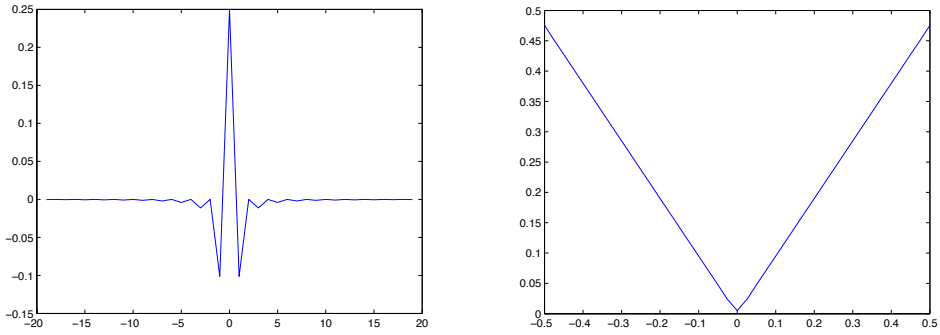


Figure 2: Illustration of the ramp filter h_{ramp}. On the left side the impulse response of the ramp filter is shown. On the right side the ideal filter response is shown in the frequency domain. It has been band-limited to $1/(2\,du)$. Here, du denotes the width of a pixel in direction $\underline{e}_u$.

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

$$\hat{f}(\underline{x}) = \frac{1}{2}\int_0^{2\pi} \mu(\lambda, \underline{x})\, g^F(\lambda, u(\lambda, \underline{x}), v(\lambda, \underline{x}))\, d\lambda\,, \tag{6}$$

where u and v are the respective detector coordinates given by

$$u(\lambda, \underline{x}) = -D\frac{\langle(\underline{x} - \underline{a}(\lambda)), \underline{e}_u(\lambda)\rangle}{\langle(\underline{x} - \underline{a}(\lambda)), \underline{e}_w(\lambda)\rangle}\,, \tag{7}$$

$$v(\lambda, \underline{x}) = -D\frac{\langle(\underline{x} - \underline{a}(\lambda)), \underline{e}_v(\lambda)\rangle}{\langle(\underline{x} - \underline{a}(\lambda)), \underline{e}_w(\lambda)\rangle}\,, \tag{8}$$

and $\mu(\lambda, \underline{x})$ is a point-dependent distance weight defined as

$$\mu(\lambda, \underline{x}) = \frac{R^2}{\langle(\underline{x} - \underline{a}(\lambda)), \underline{e}_w(\lambda)\rangle^2}\,, \tag{9}$$

which accounts for the divergent rays in a cone-beam. As usual, $\langle ., . \rangle$ denotes the standard inner product.

2.2. Implementation Approach

Despite potential deviations from an ideal source trajectory in practical cone-beam CT systems such as C-arm scanners, we perform the ramp filtering step in the direction of the detector rows. Due to the typical filter mask

sizes of 1024 and more non-zero elements, convolutions are practically computed in the Fourier domain due to the reduced computational complexity. The convolution algorithm requires the computation of the discrete Fourier transform (DFT) of each input signal (projection row) as well as the DFT of the spatial filter kernel [16]. The actual convolution of any two vectors in Fourier space is then performed by component-wise multiplication of their Fourier components. The inverse DFT (IDFT) of this product is then computed in order to transform the filtered projection rows back into the spatial domain. The input vectors are zero-padded up to a suitable power of two in order to avoid aliasing effects that may severely spoil the filtering results [17]. Since the Fourier-based convolution of two real-valued input vectors can be computed simultaneously using complex FFTs, our codes always convolve pairs of adjacent projection rows with the given filter kernel.

Due to mechanical inaccuracies of some CT devices such as mobile and stationary mounted C-arm scanners, the back-projection is commonly not implemented using Equations (6), (7), and (8) directly. Instead, the mapping from voxels in the volume to projection image positions is represented by introducing homogeneous coordinates and a corresponding 3×4 projection matrix $\check{\mathbf{P}}(\lambda)$ for each X-ray source position $\underline{a}(\lambda)$ along the scan path [18]. The projection matrices are usually estimated during a calibration step that must be performed only when the CT scanner is installed or maintained [3].

For an ideal geometry as described in Section 2.1, the projection matrices operating on points given in world coordinates can be calculated analytically from Equations (7) and (8) as follows:

$$\mathbf{P}(\lambda) = \begin{bmatrix} -D\sin\lambda & D\cos\lambda & 0 & 0 \\ 0 & 0 & D & 0 \\ -\cos\lambda & -\sin\lambda & 0 & R \end{bmatrix}. \tag{10}$$

The projection matrix can further be modified to include the transformation from voxel coordinates to world coordinates and also the transformation from detector coordinates to pixel coordinates:

$$\check{\mathbf{P}}(\lambda) = \begin{bmatrix} \frac{1}{du} & 0 & \check{u}_o \\ 0 & \frac{1}{dv} & \check{v}_o \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{P}(\lambda) \cdot \begin{bmatrix} dx & 0 & 0 & t_x \\ 0 & dy & 0 & t_y \\ 0 & 0 & dz & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11}$$

Here, du and dv denote the pixel sizes in u- and v-direction of the detector, respectively, while dx, dy, and dz denote the voxel sizes in x-, y-, and z-

direction of the WCS, respectively. Finally, t_x, t_y, and t_z represent the translation of the VCS relative to the WCS, given in world coordinates. See Figure 1 for a clarification of the symbols used in Equations (10) and (11).

Thus, back-projection can now be computed according to Algorithm 1. A matrix entry is referenced by its row and column index. Hence, $\mathbf{P}[i, j]$ refers to the entry in the ith row and the jth column of $\mathbf{P}$. Note that array data structures are assumed to be 0-based. The intermediate results $\check{u}$ and $\check{v}$ represent detector positions. For neighboring voxels, it is sufficient to increment the homogeneous detector coordinates by the appropriate column of $\check{\mathbf{P}}(\lambda)$ [3]. The homogeneous divide operation, however, cannot be avoided for voxel position increments parallel to the z-axis of the VCS because, in practical cone-beam CT systems, the projection planes are slightly tilted with respect to the z-axis due to mechanical inaccuracies. We intentionally avoided the use of a projection rebinning technique that virtually aligns the detector to one of the volume axes because it impairs the resulting image quality and requires additional computations for the initial rebinning step [12]. The actual projection value is computed by the fetch function using bilinear interpolation.

Note that, in Algorithm 1, the voxel-specific distance weight $\mu = \mu(\lambda, \underline{x})$ is determined by exploiting a computational trick. Since each projection matrix $\mathbf{P}(\lambda)$ is only defined up to a scale factor, we may normalize $\mathbf{P}(\lambda)$ such that $\mathbf{P}(\lambda)[2, 3] = 1$. In this case, it follows from Equations 9 and 10 that

$$\mu(\lambda, \underline{x}) = \frac{R^2}{\langle (\underline{x} - \underline{a}(\lambda)), \underline{e}_w(\lambda) \rangle^2} = \frac{1}{t^2}\,. \tag{12}$$

Consequently, only one additional multiply operation is necessary to compute the distance weight itself (i.e., $t^{-1} \cdot t^{-1}$) and another one to compute the weighted voxel increment $\mu \cdot \mathrm{fetch}(I_i, \check{u}, \check{v})$ afterwards.

3. Standard Multi-Core Processors

3.1. Implementation

Our implementation of the FDK processing chain is characterized by a pipeline architecture. Within this pipeline, the filtering and the back-projection are executed in dedicated stages, each using a different thread of control [19]. This enables parallel filtering of projection $n + 1$ and back-projection of projection n. In order to perform even more computations simultaneously, we further refine our implementation of both the filtering

Algorithm 1: Voxel-based back-projection.

Data: N_p projection images $I_i\,, 0 \leq i < N_p$
Input: N_p projection matrices $\check{\mathbf{P}}_i\,, 0 \leq i < N_p$
Output: volume $\mathbf{V}$ consisting of $N_x \times N_y \times N_z$ voxels

```
1  for (i = 0; i < N_p; i = i + 1) do
2      for (ž = 0; ž < N_z; ž = ž + 1) do
3          for (ў = 0; ў < N_y; ў = ў + 1) do
4              for (x̌ = 0; x̌ < N_x; x̌ = x̌ + 1) do
                   // Homogeneous image coordinates
5                  r = P̌_i[0,0] · x̌ + P̌_i[0,1] · ў + P̌_i[0,2] · ž + P̌_i[0,3];
6                  s = P̌_i[1,0] · x̌ + P̌_i[1,1] · ў + P̌_i[1,2] · ž + P̌_i[1,3];
7                  t = P̌_i[2,0] · x̌ + P̌_i[2,1] · ў + P̌_i[2,2] · ž + P̌_i[2,3];
8                  ŭ = r/t;                          // Dehomogenize
9                  v̆ = s/t;                          // Dehomogenize
10                 μ = 1/t^2;                    // Distance weight
11                 V[x̌, ў, ž] = V[x̌, ў, ž] + μ · fetch(I_i, ŭ, v̆)
12             end
13         end
14     end
15 end
```

stage and the back-projection stage to utilize the Master/Worker parallelization pattern [19]. When a projection image is processed by one of these stages, work packages are created by the master and processed by the stage's corresponding worker threads.

The convolution algorithm is implemented as described in Section 2.2. Using this approach, each image row of a projection image can be processed independently of all others. In order to efficiently compute the necessary DFTs, we use the in-place functions of Intel's *Integrated Performance Primitives (IPP)* software library for computing complex 1-D FFTs.

In our parallel implementation of voxel-based back-projection, we partition the whole volume in z-direction into sub-volumes consisting of several volume slices. Each sub-volume is assigned to one of the available worker threads. The voxels of a sub-volume are then projected onto the detector plane and updated with the corresponding interpolated projection values by

the respective worker thread. Since all worker threads have read access to the complete projection image, efficient sharing of projection data is achieved between different worker threads through the cache hierarchy of the CPU. In order to ensure that there are no competing write accesses to the volume, we process the projection images sequentially in the order they are acquired.

Today's CPUs additionally feature so-called *vector processing units (VPUs)* which are able to perform the same instruction on multiple data elements concurrently (SIMD: Single Instruction, Multiple Data). Most modern compilers can automatically vectorize code in order to take advantage of these SIMD facilities. The performance benefit of vectorization can often be further improved by manually vectorizing the code using special built-in functions, commonly referred to as *intrinsics*. Intrinsics can be thought of as substitutes for one or more inline assembly instructions. As is shown in Algorithm 1, the actual compute intensive back-projection is executed within the x-loop. This loop iterates over all voxels in one row of a slice (fixed y- and z-coordinates) and performs the same computations on each of them. We therefore concentrate on vectorizing this innermost loop by processing four consecutive voxels in x-direction simultaneously using intrinsics.

Yet, current VPUs have two inherent shortcomings. First, they cannot branch independently for individual elements of the vector and second, they cannot efficiently access memory in irregular patterns. During back-projection the access to the projection values exposes both of these problems. At the boundary of the FOV, a voxel might be projected outside of the detector while its neighbor's projection is still inside. These outliers are detected and stored in binary masks. The corresponding projection coordinates are set to $(0, 0)$ in order to avoid illegal memory accesses. Finally, the computed projection values at these positions are set to 0 to skip the voxel update. Due to the projection geometry, neighboring voxels are usually not projected onto neighboring pixels in the projection image. This results in non-contiguous memory accesses when loading the projection values. Therefore, the projection values must be read in a scalar fashion and arranged manually into SIMD vectors afterwards. Bilinear interpolation is then performed using these vectors on the VPU.

3.2. Results

In order to evaluate the performance of our implementation, we use an off-the-shelf Fujitsu Siemens workstation. It contains two Intel Xeon Quad-Core processors (E5410) running at 2.33 GHz and 16 GB RAM. A detailed

	Vectorization			
	Disabled	Compiler	Manual	Speedup
Dataset (a)	1387.0	859.8	547.9	1.6
Dataset (b)	1816.2	1135.5	722.9	1.6

Table 1: Single-threaded performance results of back-projection using the Xeon workstation.

Number of	Filtering		Back-Projection		Overall	
Threads	[s]	[pps]	[s]	[pps]	[s]	[pps]
Dataset (a)						
1	5.8	72.0	547.9	0.8	553.8	0.8
4	3.5	118.8	138.0	3.0	140.0	3.0
8	7.9	52.7	70.6	5.9	72.1	5.8
Dataset (b)						
1	14.5	37.5	722.9	0.8	736.9	0.7
4	5.6	97.9	182.0	3.0	186.0	2.9
8	6.7	81.0	93.6	5.8	96.5	5.6

Table 2: Performance results of filtering, back-projection, and both combined (overall) on the Xeon workstation using one, four, and eight cores, respectively.

description of the reconstructed datasets can be found in Appendix A.

As shown in Table 1, we achieve a substantial performance improvement of the back-projection computations by manual SIMD optimization. Our manual SIMD optimization using compiler intrinsics results in a speedup of the back-projection step by 1.6 over compiler auto-vectorization. Additionally, the performance is substantially improved by using our parallelization approach for multi-core CPUs. Table 2 shows the achieved performance for the filtering, the back-projection, and also for the overall execution of the pipeline. Note that we measure the performance of the individual algorithmic steps both in seconds (s) and in projections per second (pps).

The back-projection performance scales nearly linearly when using one, four, or eight cores of our workstation. The corresponding speedups for four and eight cores are 3.97 and 7.72, respectively. On the other hand, filtering performance is best using four cores for both datasets, although the achieved speedup does not scale with the number of cores. The performance even

degrades when using more than four cores due to synchronization overhead and the limited main memory bandwidth. Using all eight cores, our implementation is able to process about 5.6 to 5.8 projections per second when performing both filtering and back-projection (overall execution).

4. Cell Broadband Engine Architecture

4.1. Implementation

Similar to our optimized multi-core implementation we map the processing chain to a pipeline architecture in combination with the Master/Worker approach. The PPE acts as the master, dividing the processing of the stage under consideration into smaller tasks and assigning them to the available SPEs. To minimize the control overhead we assign rather large tasks to the processing elements that are further divided into smaller tasks by the processing elements themselves. Communication latencies are hidden via double buffering techniques during the dispatching and computation process.

During filtering we assign several projection rows to an SPE at the same time. The SPE then processes two rows simultaneously by loading them into its Local Store and performing the FFT-based convolution (including zero-padding).

In the back-projection step, the volume is partitioned by the PPE into relatively coarse sub-volumes to reduce communication costs. These sub-volumes are assigned to SPEs which then further divide them into small sub-volumes that fit into their Local Store, together with the corresponding projection data. 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 projection of its corner points onto the image plane, see Figure 3. We use a sub-volume size of $32 \times 32 \times 8$ voxels, which represents a reasonable trade-off between memory requirements and overall data transfer. To further reduce the communication of volume data between main memory and the SPEs, we decided to back-project at least eight projection images into the current sub-volume before storing it back to main memory and proceeding with the next sub-volume. In order to additionally accelerate data access, we store the sub-volumes sequentially in main memory. Hence, 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 (translation lookaside buffer) thrashing, we use huge pages of 16 MB size [20].

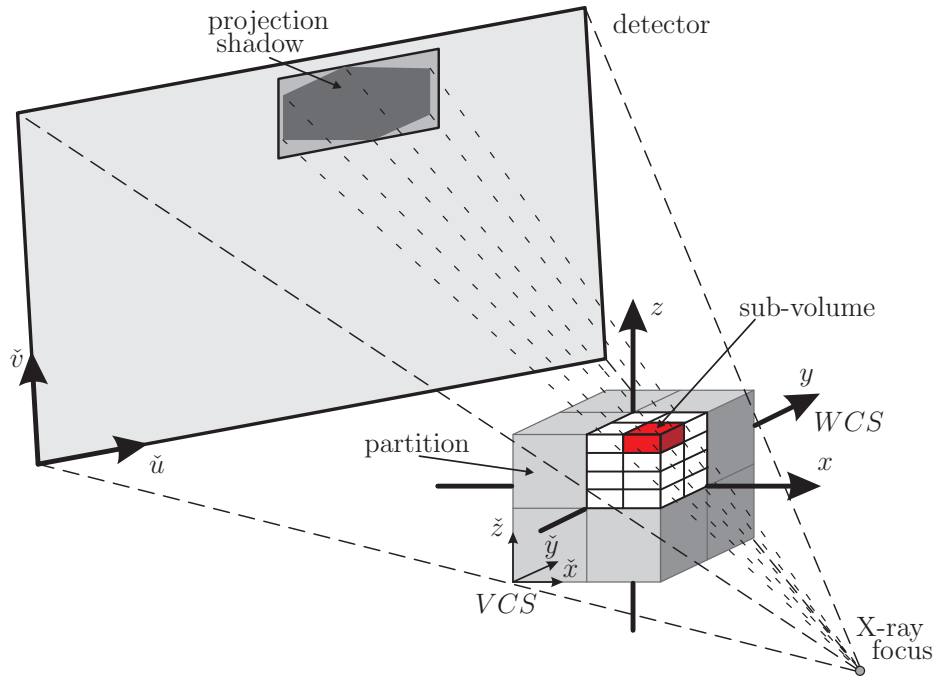


Figure 3: Parallelization strategy of our back-projection implementation using the Cell processor. Partitions are sent as tasks to the SPEs and will then be further subdivided into sub-volumes.

Each SPE can simultaneously issue (vector) instructions into two different pipelines per cycle. The even pipeline performs fixed- and floating-point arithmetic while the odd pipeline executes only load, store, and byte permutation operations. For fast implementation of an algorithm on the SPEs, their SIMD capabilities have to be exploited and, at the same time, efficient instruction scheduling to the two pipelines of each SPE must be assured. In the back-projection loop vector instructions are used for neighboring voxels in x-direction. The projection data access cannot be vectorized because the required projection values will usually not be located in consecutive and aligned memory as required by a vector operation. While the address computation and voxel update require mainly arithmetic instructions (even pipeline), the projection data access is executed on the odd pipeline. We apply loop-unrolling techniques to the iteration over the voxels of a sub-volume in order to leverage efficient instruction scheduling. SPEs cannot 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 respective slot of the destination vector.

For a bilinear interpolation operation, four projection pixel values need to be retrieved for each voxel update. This will result in poor performance unless implemented efficiently. We decrease the number of instructions for memory access on the limiting odd pipeline by adapting the data layout of the loaded projection shadow before performing the actual back-projection. More precisely, we duplicate for each projection pixel the pixel value below into the same vector. This allows to retrieve two pixel values with just one vector load instruction and thus requires only two memory accesses per voxel. This reduces the number of required instructions on the odd pipeline from 59 to 33 resulting in a speedup of 1.8 (note that only 28 arithmetic instructions are required on the even pipeline). See [21] for more details. The data layout optimization can be performed at low computational cost because it can be vectorized efficiently. The drawback of this method is that twice as much projection data must be stored in the Local Store of the SPE.

4.2. Results

The filtering and back-projection code is executed on a Blade server board covering two Cell processors running at 3.2 GHz each and 1 GB of main memory shared between the two chips. Swapping and caching strategies are

Number of SPEs	Dataset (a)			Dataset (b)		
	[s]	pps	Speed -up	[s]	pps	Speed -up
1	5.8	70.9	1.0	14.8	37.1	1.0
2	3.0	139.4	2.0	7.4	73.8	2.0
3	2.0	208.0	2.9	4.9	110.6	3.0
8	0.8	503.0	7.1	1.9	287.5	7.8
16	0.5	836.8	11.8	1.0	535.0	14.4

Table 3: Performance results of FFT-based filtering using the CBEA for the two considered datasets.

Number of SPEs	Dataset (a)			Dataset (b)		
	[s]	pps	Speed -up	[s]	pps	Speed -up
1	166.5	2.5	1.0	220.4	2.5	1.0
6	27.9	14.8	5.97	36.9	14.7	6.0
7	24.0	17.4	7.0	31.7	17.2	7.0
8	**21.0**	**19.7**	**7.9**	**27.7**	**19.6**	**8.0**
14	12.1	34.2	13.7	16.0	33.9	13.8
15	11.4	36.4	14.6	15.0	36.3	14.7
16	**10.7**	**38.8**	**15.6**	**14.0**	**38.7**	**15.7**

Table 4: Performance results of back-projection using the CBEA for the two considered datasets.

avoided by streaming the projection data either from network or hard disk while keeping the complete volume data in main memory.

Table 3 shows the respective results for the filter execution using 1, 2, 3, 8, and 16 SPEs. The speedup factor relative to the execution with only one SPE is also given, together with the number of projections that can be processed per second (pps). The FFT computations account for 90.78% of the total processing time. The data transfer time is negligible.

In Table 4 we show the achieved results when executing only the back-projection using up to 16 SPEs. The speedup factor scales almost linearly, indicating that our back-projection implementation is not affected by mem-

Number of SPEs (filtering/back-projection)		**Dataset (a)** [s]	pps	**Dataset (b)** [s]	pps
using one Cell chip	1/7	**24.0**	**17.2**	**31.9**	**17.0**
	2/6	28.6	14.8	37.0	14.7
	3/5	33.5	12.4	44.3	12.3
using two Cell chips	1/15	**11.4**	**36.2**	16.2	33.5
	2/14	12.2	34.0	**16.1**	**33.8**
	3/13	13.1	31.6	17.2	31.5

Table 5: Overall pipelined execution of filtering and back-projection for the two considered datasets, bold numbers refer to optimum configurations.

ory bandwidth limitations.

We finally execute the filtering and the back-projection in parallel in a pipeline software architecture. This approach has the huge advantage that it does not require much main memory capacity to temporarily save the filtered projection data. The number of SPEs for filtering and back-projection, however, has to be chosen statically before execution. Table 5 shows the corresponding results for various configurations. When using only eight SPEs, it is sufficient to perform the filtering with one SPE. The measured overall runtime is close to the measured execution time of the back-projection alone using seven SPEs. Thus, filtering execution is fully hidden behind the back-projection. It can also be seen that more than 30 projections can be reconstructed per second with two Cell processors, which is sufficient for real-time on-the-fly reconstruction that is synchronized with the acquisition process.

5. Graphics Accelerator Boards

5.1. Implementation

Again, the processing chain of the FDK algorithm is mapped to a pipeline architecture. In addition to the stages for filtering and back-projection we introduce a projection upload stage and a volume download stage. The device memory for projections is allocated inside the projection upload stage, while the device memory for the volume is allocated inside the back-projection stage. The stages are then connected single-threaded in order to share the same CUDA device context.

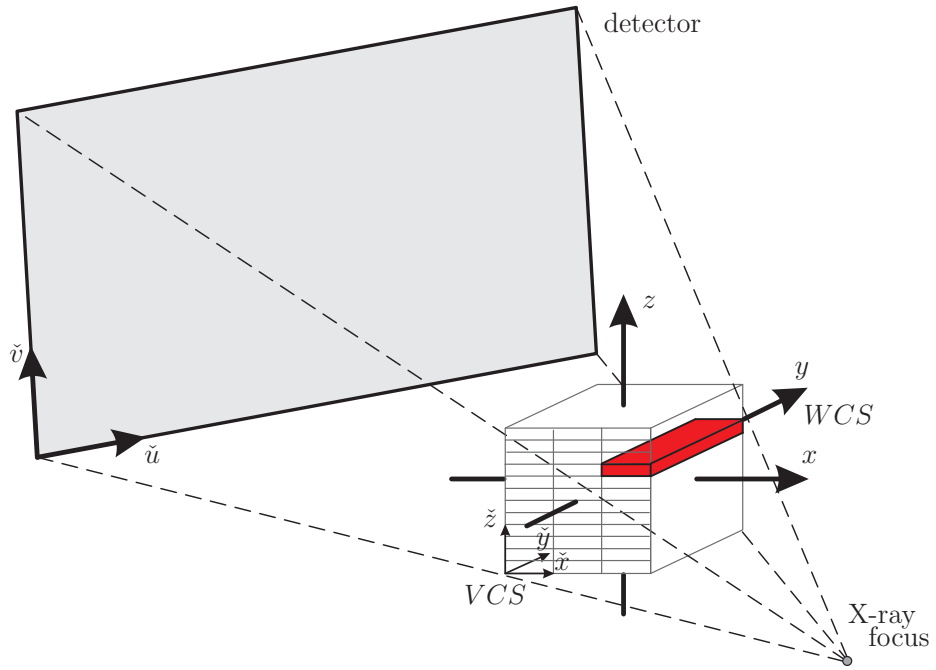


Figure 4: Parallelization strategy of our back-projection implementation on the GPU using CUDA. The $\check{x}$-$\check{z}$ plane is divided into several blocks to specify a grid configuration, and each thread of a corresponding block processes all voxels in $\check{y}$-direction.

The filtering stage is implemented using the *CUFFT* library of the CUDA package. All additional required computations are mapped to several successive CUDA kernel executions. Each CUDA kernel computes all rows of a complete projection image simultaneously in order to efficiently exploit the multiprocessors on the graphics card.

For the computation of the voxel-based back-projection on the GPU, we store the complete volume in global device memory. We further bind a texture context to the projection data. Texture memory is not only optimized for fast random memory accesses. Rather, it further enables a performance improvement by using the texture hardware of the GPU for hardware-accelerated bilinear interpolation. The corresponding projection matrix is stored in constant device memory. Then we invoke our back-projection kernel on the graphics device. Each thread of the kernel computes the back-projection for all voxels of a certain column in a volume slice, see Figure 4. The increased number of registers on NVIDIA GPUs of the 200 series (e.g., GeForce GTX 280, Quadro FX 5800, and Tesla C1060) allows to loop over

GPU	GeForce 8800 GTX	Tesla C870	GeForce GTX 280	Tesla C1060
Architecture	8 Series		200 Series	
Processor Cores	128		240	
Processor Clock [MHz]	1350		1296	
Gflops [MADD/MUL]	518		933	
Bus Width [bit]	384		512	
Memory Clock [MHz]	900	800	1107	800
Memory Bandwidth [GB/s]	86.4	76.8	141.7	102.4
Memory [GB]	0.768	1.500	1.000	4.000

Table 6: Technical overview of the considered graphics accelerator boards from Nvidia. The Tesla C870 is identical to the Quadro FX 5600. Likewise is the Tesla C1060 identical to the Quadro FX 5800. The difference between Quadro and Tesla is only in reliability of the memory and OpenGL support.

a few projection images in the inner-most loop. Using this approach, a large portion of previously required global memory accesses can be avoided. For example, looping over two projections in the inner-most loop requires only half as many global memory accesses than processing just a single projection.

In order to achieve high memory bandwidth on GPUs of the 200 series, the volume memory layout has to be padded appropriately. The memory architecture of these devices is taken into account by padding each row such that its total size in bytes is divisible by 32, but not by 512 (e.g. 32 bytes padding for a 512^3 volume). In the following, we will refer to the adaption of the memory layout as *address aliasing fix (aaf)*. This is necessary on GPUs of the 200 series in order to avoid a drastic reduction in device memory bandwidth.

5.2. Results

We evaluated the filtering and back-projection performance of our GPU implementation using six different graphics accelerators from NVIDIA (see Table 6). It is crucial to choose an appropriate grid configuration which organizes many lightweight CUDA threads in a two-level hierarchy: a grid, which consists of one or more blocks where each block comprises a specific number of threads. The grid configuration influences both the global memory access pattern and the texture cache usage. Our experiments reveal that it

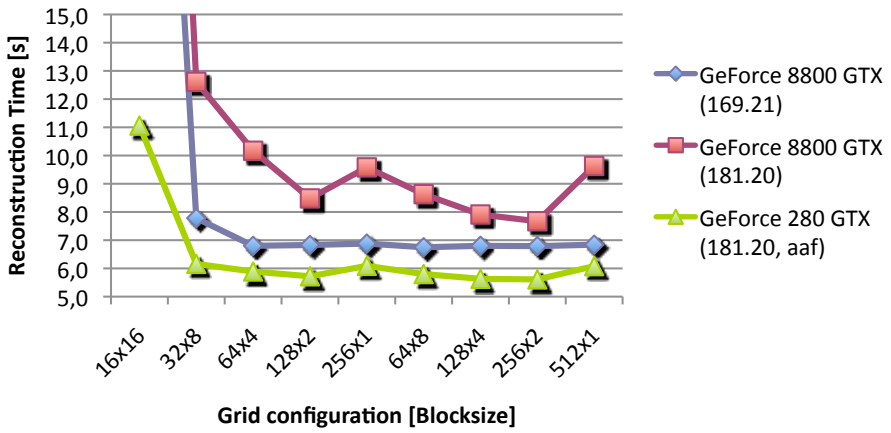


Figure 5: GPU execution times for different grid configurations and different graphics driver versions.

is more important to optimize the global memory accesses than the texture cache usage, see Figure 5. Note that it is essential to have at least 64 to 128 threads in a block in x-direction, otherwise reconstruction performance is significantly degraded. This is more important on GPUs of the 8 series (e.g., GeForce 8800 GTX, Quadro FX 5600, and Tesla C870), as can be seen in Figure 5 as well. For both device series, a grid configuration of 256×2 is demonstrated to be optimal. We have made the observations that different GPU driver versions may lead to significantly varying performance results. This needs to be kept in mind when comparing GPU-based performance data.

In the next step we fix the grid configuration and evaluate how the back-projection performance is influenced when looping over a few projection images in the inner-most loop. Figure 6 shows the achieved performance results. Unfortunately, NVIDIA GPUs of the 8 series cannot execute any back-projection kernel with an unrolled inner-most loop due to their limited number of GPU registers and even on GPUs of the 200 series the projection loop was limited to two projections when using 3-D textures and to three projections when using flattened 2-D textures. In this regard, approaches which use projection rebinning techniques [12] can further improve the reconstruction speed since they require fewer computations and projection matrix accesses during back-projection. However, we intentionally avoid rectification-based approaches. It can further be seen in Figure 6 that the

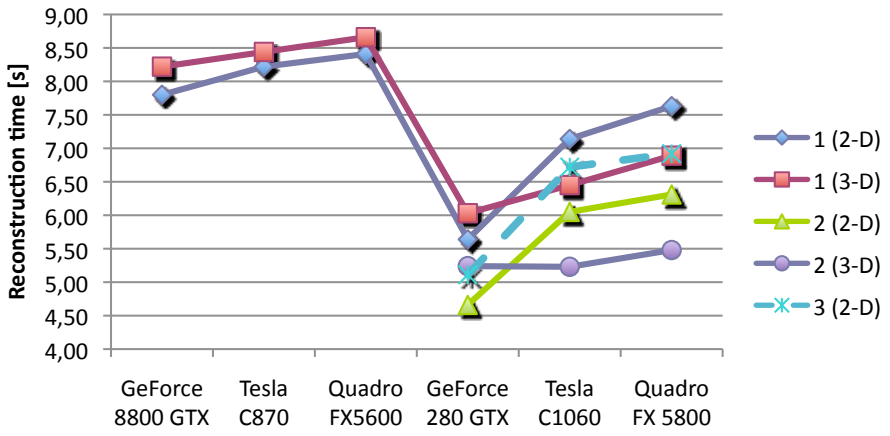


Figure 6: Comparison of back-projection performance on the GPU when using a loop over several projections in the inner-most back-projection loop.

graphics driver follows different driver paths for each branch of devices. For example, it is more efficient to save all projections in one flattened 2-D texture than in a 3-D texture using the GeForce devices. Using Quadro or Tesla devices, however, it is more efficient to use a 3-D texture. Please note that it is also not an alternative to use several 2-D texture references because this results in a much higher register usage on all considered devices and degrades performance significantly.

Unrolling the inner-most loop over two projection images proves to be a very efficient optimization technique. The bandwidth limitation of back-projection is reduced due to the saved global memory accesses.

In Table 7, we show the timing measurements for the filtering, back-projection, and also for the overall execution for each device under consideration. We also give the numbers of projections that can be processed per second (pps). Comparing the measured results of the Tesla and Quadro devices, we achieve slightly better reconstruction speed for the Tesla devices, although Tesla and Quadro devices of the same GPU series are based upon the same hardware components. This difference seems to be based on the overheads caused by frequent thread context changes on the Quadro devices since we also use the Quadro device as the primary display adapter in our test systems. The FFT-based filtering is not affected by memory bandwidth, and between GPUs of the 8 series and the 200 series the theoretical speedup of 1.8

Hardware	Filtering		Back-Projection		Overall	
	[s]	pps	[s]	pps	[s]	pps
Dataset (a)						
Quadro FX 5600	2.6	158.6	8.3	50.2	12.0	34.4
Tesla C870	2.3	181.6	8.1	51.4	11.4	36.3
GeForce 8800 GTX	2.2	184.8	7.5	55.0	10.7	38.6
Quadro FX 5800	1.6	265.4	5.4	76.4	7.9	52.3
Tesla C1060	1.3	309.0	5.2	79.3	7.3	57.1
GeForce GTX 280	1.5	281.6	4.6	89.2	7.1	58.5
Dataset (b)						
Quadro FX 5600	6.6	82.9	11.5	47.3	19.8	27.5
Tesla C870	6.0	91.0	11.2	48.5	18.5	29.4
GeForce 8800 GTX	5.9	91.9	10.6	51.3	17.8	30.5
Quadro FX 5800	3.9	138.5	8.5	63.9	13.5	40.2
Tesla C1060	3.5	156.5	8.2	66.1	12.6	43.1
GeForce GTX 280	3.6	153.0	6.7	80.7	11.4	47.5

Table 7: GPU performance results of filtering and back-projection for the considered devices of the 8 series and 200 series.

due to computing resources is nearly reached. This is, however, not the case for the back-projection. The difference in execution times for the Quadro and GeForce follows roughly the difference in device memory bandwidth when no loop unrolling is applied. This indicates that our back-projection performance is bandwidth-limited. In this regard, the loop over multiple projections in the innermost back-projection loop proves to be a very effective optimization. Thus, the increase in performance between the two considered GPU series does not follow the speedup in device memory bandwidth anymore. Instead the reached speedup factor is right in the middle between the speedup that is possible due to the increased amount of computing resources and higher memory bandwidth.

6. FPGA-Based Hardware

Generally speaking, an FPGA is a dynamically reconfigurable microchip that covers logical hardware blocks (e.g., look-up tables), arithmetic units (e.g., multiply-add blocks), as well as I/O functionality [2]. Todays FPGA designs typically run at clock rates of 100 MHz up to 500 MHz. As an example of an FPGA-based accelerator hardware, we focus on the *ImageProX (image processing accelerator)* board that has been developed at Siemens Healthcare and was released in 2006. The ImageProX board uses either a 64 bit PCI interface (66 MHz) or a 64 bit PCI-X interface (133 MHz) to connect to the host PC. It covers nine Xilinx Virtex-4 FPGAs (1× Virtex-4 SX55, 8× Virtex-4 SX35), each of which is equipped with up to 1 GB of external DDR2 SDRAM memory. The ImageProX board comprises two identically organized rings of four Virtex-4 SX35 chips each, with the even more powerful Virtex-4 SX55 FPGA representing the core control and interface unit of the design [8]. Note that FPGAs currently offer fixed-point arithmetic only. However, due to the efficient implementation of pseudo floating-point arithmetic, the image quality delivered by our ImageProX-based FDK implementation is comparable to the image quality of the other platforms under consideration for clinically relevant data.

6.1. Implementation

The filter stage of our ImageProX-based implementation is completely implemented as part of the Virtex-4 SX55 firmware and runs fully pipelined at a clock rate of 200 MHz. Since two projection rows are filtered simultaneously, this yields a processing speed of 400 Mega samples per second. With

each sample being represented as a 16 bit fixed-point value, the filter stage is thus able to process 800 MByte of projection data per second, which exceeds the peak bandwidth of the 66 MHz PCI interface and is close to the peak bandwidth of about 1 GByte per second of the 133 MHz PCI-X interface. Consequently, the filtering stage does not represent a data processing bottleneck. The FFT/IFFT blocks of the ImageProX filtering stage are generated using the Xilinx CoreGen library. Internally, a block floating-point format is employed in order to achieve a significantly high numerical accuracy of the filtering results.

The back-projection is accomplished simultaneously by the eight Virtex-4 SX35 FPGAs. The volume is being reconstructed in a blockwise manner. In a typical reconstruction setting with nearly optimal load balancing, each of the SX35 chips will store approximately the same number of projection images in its external SDRAM memory. Hence, each of the SX35 chips is responsible for back-projecting its own set of projection images into the current volume block. Analogous to the filtering engine, the back-projection design is fully pipelined as well, such that a peak speed of about 25 Giga back-projection steps per second can be achieved in theory: 200 MHz $\times$ 8 SX35 FPGAs $\times$ 16 back-projection pipelines per SX35 FPGA. Within the SX35 chip 16 so-called *parallel back-projection units (PBUs)* simultaneously read a portion of the FPGA-internal memory that stores the portion of the current projection to be back-projected into the volume. The current volume block to be written is kept in a separate portion of FPGA-internal memory. It is important to point out that the back-projection phase can only start as soon as all projections (or at least a significantly large portion of projections) have been transferred into the FPGA-external memories. An on-the-fly reconstruction that immediately processes and back-projects a projection as soon as it becomes available is therefore not possible using this ImageProX implementation [8].

6.2. Results

Table 8 shows, among others, the reconstruction performance of a PCI-based ImageProX accelerator board. The comparison of the back-projection speed and the overall reconstruction speed reveals that the reconstruction speed of this platform is limited by the bandwidth of the PCI bus. In theory, the PCI-X bandwidth is twice as high as the PCI bandwidth such that a correspondingly higher overall performance will result as soon as the PCI-X implementation is used instead.

Hardware	Filtering		Back-Projection		Overall	
	[s]	pps	[s]	pps	[s]	pps
Dataset (a)						
CPU	3.5	118.8	138.0	3.0	140.0	3.0
CBEA	0.8	503.0	21.0	19.7	24.0	17.2
GPU	1.3	309.0	5.2	79.3	7.3	57.1
FPGA			3.9	107.3	11.2	36.9
Dataset (b)						
CPU	5.6	97.9	182.0	3.0	186.0	2.9
CBEA	1.9	287.5	27.7	19.6	31.9	17.0
GPU	3.5	156.5	8.2	66.1	12.6	43.1
FPGA			4.4	124.5	15.0	36.3

Table 8: Reconstruction performance results of the FDK method (filtering and back-projection) for all hardware alternatives under consideration. Performance measurements of an Intel quad-core processor running at 2.3 GHz (CPU), of a Cell processor with eight SPEs running at 3.2 GHz, of an NVIDIA Tesla C1060 computing accelerator (GPU), and of the ImageProX accelerator board (FPGA) are shown. Note that, due to the highly integrated design of the ImageProX implementation, the FPGA-specific filtering performance cannot be given separately.

7. Discussion

Table 8 shows a comparison of the achievable reconstruction speed for the FDK method using the previously described target architectures. In order to compute an FDK reconstruction on-the-fly (i.e., in real-time) using a current practical cone-beam CT scanner, at least 30 projections per second must be processed. It can be seen from Table 8 that current GPU devices are ideal candidates when reconstructions shall be computed on-the-fly while projection data is being acquired. A real-time reconstruction can also be achieved with two Cell processors. Yet, it is difficult to find cost-efficient commercial solutions that offer two Cell processors on a single mainboard. As far as we know, there are currently only Cell Blades commercially available that provide two Cell processors acting as an SMP machine. Using the ImageProX FPGA board, it is only possible to upload and filter the projection images on-the-fly, since our implementation requires all projection images in order to compute the back-projection result. However, the back-

projection computation using the ImageProX accelerator takes less than five seconds for each of the datasets given in Table 8. Interestingly, in comparison to the other considered hardware architectures, the ImageProX accelerator delivers the fastest back-projection performance. On the other hand, it is nearly impossible to build systems that are able to accomplish on-the-fly reconstructions using only a few cores of current general-purpose processors (e.g., Intel- or AMD-based). Their reconstruction performance is far from the speed exposed by the more specialized architectures under consideration. The achieved results demonstrate that a performance increase of an order of magnitude and even more is achievable compared to recent high-performance general-purpose computing platforms, see again Table 8. With more than 40 projections per second, high-end graphics accelerators currently deliver the highest overall reconstruction speed.

While implementation complexity typically tends to be comparatively low for CPU-based systems, highly optimized CPU implementations are becoming as complicated as implementations for more specialized architectures like Cell processor based systems and FPGA accelerators. According to our experience, GPU systems provide a reasonable balance between implementation effort and achievable reconstruction speed.

A downside of the GPUs is their co-processor based architecture. A corresponding CPU core has to be present in order to control and to synchronize the GPU calculations. We observed high CPU load during GPU computations indicating that synchronization is presently still being accomplished using busy-waiting loops.

8. Conclusions

We have presented highly optimized implementations of the FDK method for four different state-of-the-art hardware architectures and evaluated their reconstruction performance using two medical datasets that were acquired using a standard C-arm device.

It is highly difficult—if not impossible—to objectively compare the reconstruction speed of different hardware architectures as long as not the same preconditions are fulfilled. Linear scaling and comparison of published results may lead to wrong conclusions concerning the achievable reconstruction speed due to the use of different hardware, different datasets, different reconstruction parameters, and due to many implementations that assume

an acquisition with ideal geometry, which is unfortunately not the case in most practical CT scanners.

The results demonstrate that a performance increase of more than an order of a magnitude is achievable compared to recent multi-core CPUs. High-end graphics accelerators currently deliver the highest overall reconstruction speeds. This makes them especially well suited for the real-time computation of cone-beam CT reconstructions, meaning that all required computations can be concealed by the scan time of the X-ray acquisition device.

9. Acknowledgments

This work was supported by Siemens Healthcare, CV Division, Medical Electronics and Imaging Solutions.

Appendix A. Datasets

Dataset (a) consists of 414 projection images of 1024×1024 pixels each. The convolution length for the FFT-based filtering step is 2048 (including zero-padding). The isotropic voxel size of the 512^3 volume is set to $(0.26\ \text{mm})^3$ such that the entire volume is located inside the FOV. The reconstructed volume shows a human head.

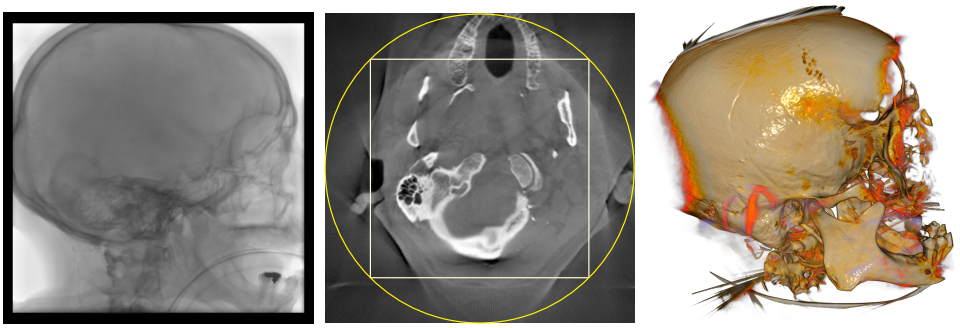
Dataset (b) consists of 543 projection images of 1240×960 pixels each. The convolution length is 4096 (including zero-padding), and the isotropic voxel size of the 512^3 volume is set to $(0.31\ \text{mm})^3$. The reconstructed volume shows a phantom of a human hip.

The used data types for both preprocessed projections and reconstructions are single precision floating-point (32-bit). Only the FPGA-based implementation uses an efficient implementation of pseudo floating-point arithmetic as described in Section 6.

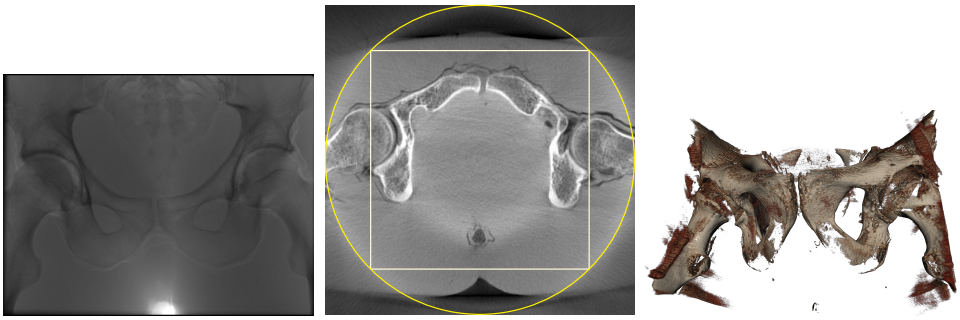
Images of both datasets are shown in Figure A.7.

References

[1] L. Feldkamp, L. Davis, J. Kress, Practical cone-beam algorithm, Journal of the Optical Society of America A1 (6) (1984) 612–619.



Dataset (a).



Dataset (b).

Figure A.7: Image results of the two datasets which have been used for the performance measurements are shown (Dataset (a) in top row and Dataset (b) in bottom row). From left to right a single projection image, a reconstructed slice image, and a 3-D image computed by a volume rendering technique (VRT) are shown, respectively. The projection images of these datasets show a boundary region of zero values. During reconstruction these borders have been assumed to contain image information. The slice images have been reconstructed using our GPU implementation and are scaled to their full intensity range each. The yellow circles indicate the FOV, while the white squares indicate the regions, which have been reconstructed during the performance evaluations of the FDK method.

[2] U. Meyer-Bäse, Digital Signal Processing with Field Programmable Gate Arrays (Signals and Communication Technology), 2nd Edition, Springer, 2008.

[3] K. Wiesent, K. Barth, N. Navab, P. Durlak, T. Brunner, O. Schuetz, W. Seissler, Enhanced 3-D-reconstruction algorithm for C-arm systems suitable for interventional procedures, IEEE Transactions on Medical Imaging 19 (5) (2000) 391–403.

[4] R. Yu, R. Ning, B. Chen, High-speed cone-beam reconstruction on PC, in: Proc. SPIE Medical Imaging 2001: Image Processing, Vol. 4322, San Diego, USA, 2001, pp. 964–973.

[5] M. Kachelrieß, M. Knaup, O. Bockenbach, Hyperfast perspective cone-beam backprojection, in: IEEE Nuclear Science Symposium and Medical Imaging Conference, San Diego, USA, 2006.

[6] M. Trepanier, I. Goddard, Adjunct processors in embedded medical imaging systems, in: Proc. SPIE Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display, Vol. 4681, 2002, pp. 416–424.

[7] I. Goddard, M. Trepanier, High-speed cone-beam reconstruction: an embedded systems approach, in: Proc. SPIE Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display, 2002, pp. 483–491.

[8] B. Heigl, M. Kowarschik, High-speed reconstruction for C-arm computed tomography, in: Proc. Int. Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine (Fully 3D) and Workshop on High-Performance Image Reconstruction (HPIR), Lindau, Germany, 2007, pp. 25–28.

[9] M. Kachelrieß, M. Knaup, O. Bockenbach, Hyperfast parallel-beam and cone-beam backprojection using the Cell general purpose hardware, Medical Physics 34 (4) (2007) 1474–1486.

[10] G. Yan, S. Zhu, Y. Dai, C. Qin, Fast cone-beam CT image reconstruction using GPU hardware, Journal of X-Ray Science and Technology 16 (2008) 225–234.

[11] F. Xu, K. Mueller, Real-time 3D computed tomographic reconstruction using commodity graphics hardware, Physics in Medicine and Biology 52 (2007) 3405–3419.

[12] C. Riddell, Y. Trousset, Rectification for cone-beam projection and backprojection, IEEE Transactions on Medical Imaging 25 (7) (2006) 950–962.

[13] K. Mueller, F. Xu, N. Neophytou, Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?, in: SPIE Electronic Imaging Conf., San Diego, USA, 2007.

[14] M. Churchill, Hardware-accelerated cone-beam reconstruction on a mobile C-arm, in: Proceedings of SPIE, Vol. 6510, 2007.

[15] T. M. Buzug, Computed Tomography: From Photon Statistics to Modern Cone-Beam CT, Springer-Verlag, Berlin/Heidelberg, 2008.
URL http://www.springer.com/medicine/radiology/book/978-3-540-39407-5

[16] A. Kak, M. Slaney, Principles of Computerized Tomographic Imaging, SIAM, 2001.

[17] R. Gonzalez, R. Woods, Digital Image Processing, 3rd Edition, Pearson Education, Inc., 2008.

[18] R. Hartley, A. Zissermann, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, Cambridge, UK, 2003.

[19] H. Scherl, S. Hoppe, M. Kowarschik, J. Hornegger, Design and implementation of the software architecture for a 3-D reconstruction system in medical imaging, in: ICSE '08: Proc. 30th Int. Conference on Software Engineering, New York, USA, 2008, pp. 661–668.

[20] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, 3rd Edition, Morgan Kaufmann, 2003.

[21] H. Scherl, M. Koerner, H. Hofmann, W. Eckert, M. Kowarschik, J. Hornegger, Implementation of the FDK algorithm for cone-beam CT on the Cell Broadband Engine Architecture, in: Medical Imaging 2007: Physics of Medical Imaging, Vol. 6510 of Proceedings of SPIE, 2007, p. 651058.