A GPU-based Architecture for Real-Time Data Assessment at Synchrotron Experiments

Suren Chilingaryan, Alessandro Mirone, Andrew Hammersley, Claudio Ferrero, Lukas Helfen, Andreas Kopmann, Tomy dos Santos Rolo, Patrik Vagovic

***Abstract*—Advances in digital detector technology leads presently to rapidly increasing data rates in imaging experiments. Using fast two-dimensional detectors in computed tomography, the data acquisition can be much faster than the reconstruction if no adequate measures are taken, especially when a high photon flux at synchrotron sources is used. We have optimized the reconstruction software employed at the micro-tomography beamlines of our synchrotron facilities to use the computational power of modern graphic cards. The main paradigm of our approach is the full utilization of all system resources. We use a pipelined architecture, where the GPUs are used as compute coprocessors to reconstruct slices, while the CPUs are preparing the next ones. Special attention is devoted to minimize data transfers between the host and GPU memory and to execute memory transfers in parallel with the computations. We were able to reduce the reconstruction time by a factor 30 and process a typical data set of 20 GB in 40 seconds. The time needed for the first evaluation of the reconstructed sample is reduced significantly and quasi real-time visualization is now possible.**

***Index Terms*—Synchrotrons, Computed tomography, Image reconstruction, Software, High performance computing, Parallel programming, GPU computing, Performance evaluation.**

I. Introduction

DRIVEN by substantial advances in digital detector technology, there is presently a rapid progress in X-ray imaging technologies opening many applications in the fields of medical diagnostics, homeland security, non-destructive testing, materials research and others. X-ray imaging permits spatially resolved visualization of 2D and 3D structures in materials and organisms which is crucial for the understanding of their properties. Furthermore, it allows one to recognize defects in devices from the macro- down to the nano-scale. Additional resolution in the time domain gives insight into the dynamics of processes allowing one to understand the functionality of organisms and to optimize devices and technological processes.

In recent years, synchrotron tomography has seen a substantial decrease of scan durations [1]. Based on the available photon flux densities at modern synchrotron sources, ultra-fast X-ray imaging enables the investigation of the dynamics of technological and biological processes with a time scale down to the milliseconds range in 3D. Using modern CMOS-based pixel cameras, it is possible to reach image rates of up to several thousand frames per second. For example, frame rates of 5000 images per second were achieved [2] using a filtered white beam from the ESRF (European Synchrotron Radiation Facility) ID19 wiggler source, a frame rate of 40000 images per second were reported [3] using different experimental conditions at ESRF beamline ID15a and a larger effective pixel size. As a result of the improved image acquisition, a given experiment can produce data sets of multiple gigabytes in a few seconds. It is a major challenge to process the data in a reasonable amount of time facilitating on-line reconstruction.

Several approaches are currently used to handle the huge data sets produced at the synchrotron imaging beamlines.

- At the *TopoTomo* beamline of ANKA (the synchrotron facility at the Karlsruhe Institute of Technology [4]) the data has been stored in a local memory, transferred to mass storage, and then processed and analyzed off-line. The data quality and thus the success of the experiment could only be judged with a substantial delay, which made an immediate monitoring of the results impossible.
- At the ESRF, the experiments are usually monitored by distributing the reconstruction of 3D volume onto different hosts in a cluster via a queuing system. This approach was adopted to maximize overall throughput rather than minimizing reconstruction time for a single scan.
- A pipelined data acquisition system combining a fast detector system, high speed data networks, and massively parallel computers is employed at the APS (Advanced Photon Source at the Argonne National Laboratory) to acquire and reconstruct a full tomogram in tens of minutes [1]. At the Paul Scherrer Institute in Switzerland, this approach was further improved, reducing the reconstruction time down to just a few minutes [5]. However, the supercomputer-based processing is expensive in terms of money, power consumption, and administrative effort.

Our approach exploits the computational power of modern graphic adapters which include hundreds of simple processors to transform vertices in 3D space. These processors can be used to speed up the reconstruction. The peak performance of the fastest GPUs exceeds 1 TFlop/s for single precision operations. High-end gaming desktops include up to four cards and provide almost 5 TFlop/s of computational power. As compared to 100 GFlop/s provided by commonly used servers, this gives a potential speedup of a factor 50 [6], [7].

Manuscript received June 30, 2010; revised November 19, 2010; revised February 22, 2011

S. Chilingaryan and A. Kopmann are with Institute for Data Processing and Electronics, Karlsruhe Institute of Technology, Karlsruhe, Germany (telephone: +49 724 7826579, e-mail: Suren.Chilingaryan@kit.edu).

A. Mirone, A. Hammersley, and C. Ferrero are with European Synchrotron Radiation Facility, Grenoble, France.

L. Helfen, T. dos Santos Rolo, and P. Vagovic are with Institute for Synchrotron Radiation, Karlsruhe Institute of Technology, Karlsruhe, Germany.

Since the release of the CUDA (Common Unified Device Architecture [8]) toolkit, this computational power can be used by developers to carry out general purpose computations. CUDA extends the C language by a number of syntactical constructs to enable data parallel computations on GPUs. The architecture allows one to execute a sequence of mathematical operations, so called kernels, simultaneously over a multi-dimensional set of data. The API (Application Programming Interface) includes functions transferring the data between the system and GPU memory and executing the kernels to process the data. For developers convenience NVIDIA has released a number of libraries implementing standard algorithms. The CUDA SDK includes cuFFT - an interface to perform multi-dimensional FFT (Fast Fourier Transformations), cuBLAS - a BLAS (Basic Linear Algebra Subprograms) implementation, and cudpp - a collection of parallel reduction algorithms. There are third-party implementations of LAPACK (Linear Algebra PACKage) and a number of computer vision algorithms [9], [10], [11].

In addition to vendor-dependent programming toolkits like CUDA, the Khronos Group has introduced OpenCL (Open Computing Language [12]), a new industry standard for parallel programming. The architecture of OpenCL is similar to CUDA, but, unlike the proprietary NVIDIA technology, it allows one to execute the developed applications on both AMD and NVIDIA GPUs as well as on general-purpose CPUs.

To reconstruct 3D images of objects from multiple projections the FBP (Filtered Back Projection) algorithm is frequently used [13], [14]. A fast implementation of this algorithm is *PyHST* (High Speed Tomography in Python [15]), which was developed at the ESRF. It is currently used at the ESRF, ANKA, and BESSY synchrotron facilities. While it uses heavily optimized *C*-code for performance critical parts of the algorithm, the processing of a typical data set with *Core* and *Nehalem* based Xeon servers still require about one hour of computations. This time is made up from the I/O operations dealing with image files and the computations of the FBP reconstruction in approximately equal parts. In this work we present a solution which reduces the computation time to the minute scale and, hence, solves the computational part of the problem. The options to transfer the image files in a faster way are discussed as well.

In order to improve the performance and achieve a near real-time reconstruction speed, we accelerated the original version of *PyHST* by shifting most of the computations to the GPU using the CUDA toolkit. The optimized version is able to reconstruct a typical 3D image (a 3 gigavoxel image reconstructed from 2000 projections) in only 40 seconds using a GPU server equipped with 4 graphic cards, I/O time excluded. It is approximately 30 times faster compared to the 8-core Xeon server used before.

There are a few other research projects aiming for GPU-assisted tomographic reconstruction. A group at the University of Antwerp has assembled a GPU server running 7 NVIDIA GeForce GTX 295 cards with a total peak performance of 12 TFlop/s. The presented benchmark shows a 35 times speedup compared with Intel Core i7-940 [16]. RapidCT is another project developing GPU-assisted tomographic software [17]. The presented results indicate an increase of execution speed by a factor 20. Earlier implementations of the filtered back projection were developed using the shader language [18], [19]. Unfortunately, the projects do not distribute the developed software and we are not able to compare the achieved performance with our results directly.

The article is organized as follows. Section II describes the FBP algorithm and provides an estimation of the computational complexity. Section III gives details about *PyHST* and our optimizations. The performance evaluation and a review of available hardware platforms are provided in Section IV. The I/O bottleneck is discussed in Section V. Finally, the project highlights are summarized.

II. Tomography at Synchrotron Light Sources

Due to rather large source-to-sample distances, imaging at synchrotron light sources is usually well described by a parallel-beam geometry. The sample is placed onto a rotation stage in front of a pixel detector and rotated in equiangular steps. The X-rays penetrate the sample and the pixel detector registers a series of parallel 2D projections of the sample volume. If the projection of the rotation axis onto the detector plane is aligned perpendicular to the lines of the pixel detector and the beam direction is also perpendicular to the rotation axis, the 3D reconstruction problem can be split up into a series of 2D reconstructions performed with cross-sectional slices. The projection information needed for the reconstruction of a given slice can always be extracted from the same detector line of the image series. A coordinate system is defined so that the center of rotation is located at the center of the sample coordinate system and the sample is rotating around the vertical axis. The principle of back projection is used for the reconstruction of the slices. To determine a function of the sample object at a given position with coordinates (x, y) in the slice z, it is necessary to compute $\sum_{p=1}^{P} I_p(x \cdot \cos(p\alpha) - y \cdot \sin(p\alpha), z)$ over all projections, where P is the number of projections, α is the angle between projections, and I_p is the image of p-th projection. For a given slice, this corresponds to smearing back the projection values over the 2D cross section, and integrating over all projection angles, see Fig. 1. Since the images are digitized, a linear interpolation is performed to get the values of I_p at the computation point [13], [20].

This approach, however, yields blurred results. As it can be seen from Fig. 1, for a low number of projections the image is affected by streaks along the back projection directions. Moreover, the image reconstructed by simple (i.e. unfiltered) back projection contains strong low-frequency components, often visible as an envelope across the image. High-pass filtering of the projection data prior to back projection is used to compensate the blurring effect (e.g. by multiplication in the Fourier domain with a ramp filter [13] or other filter types [21]). Fig. 2 illustrates the difference between the reconstructed images using unfiltered and filtered back projections. If filtering is not performed, the over-represented low frequencies blur important details in the reconstructed image [22].

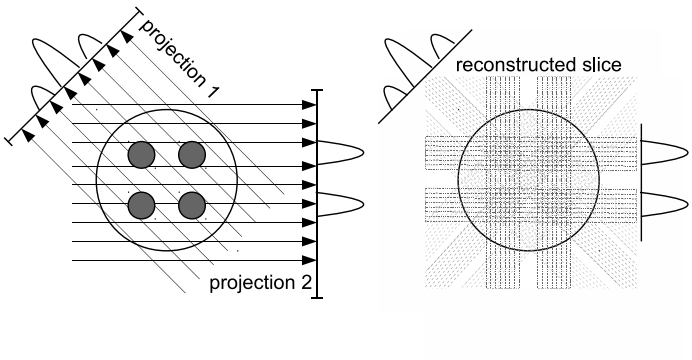

Fig. 1. Image reconstruction using a simple back projection. The sample is rotated and a pixel detector acquires a series of parallel projections under different rotation angles (left). For image reconstruction, all projections are smeared back onto the cross section along the direction of incidence yielding an integrated image (right).

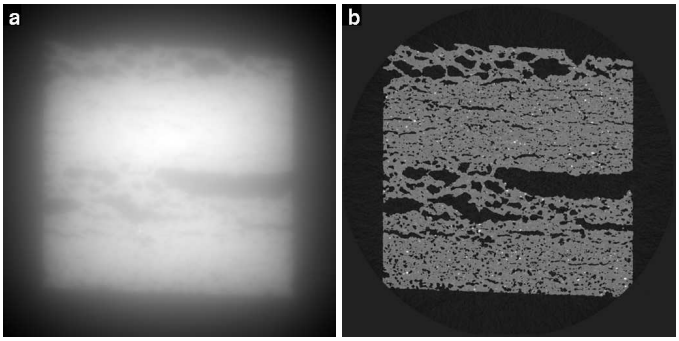

Fig. 2. Aluminum alloy foams are created by expanding foamable precursors containing a gas-releasing blowing agent in a dense metallic matrix. The figure visualizes an aluminum foam at an early expansion stage obtained by synchrotron radiation based micro-tomography. No filtering was performed for image **a** while a linear ramp filter was used to reconstruct image **b**.

A. Algorithm Complexity

The source data includes S slices consisting of P projections and N bins in each projection. The reconstructed 3D image must be composed of M voxels totally. Both the source and the result are stored uncompressed in a single-precision floating point format (32 bit). If linear interpolation is used the total number of operations required to do the back projection is equal to $9 \cdot P \cdot M$ [23]. In order to filter a single projection, a convolution must be carried out for each slice individually. The complexity of the slice filtering is equal to the complexities of the direct and inverse FFT transformations and the multiplication with the filter in a Fourier domain. Assuming that the number of elements in the transformed vector (N) is equal to a power of 2, the exact number of floating point operations required to compute FFT is $N \cdot (a \cdot \log_2 N + 6)$, where a is between 4.03 and 5 depending on the radix of the algorithm [24]. For sake of simplicity $N \cdot (5 \cdot \log_2 N + 6)$ is used to estimate the computational complexity of the FFT. Then, the total number of operations required to filter a single slice is equal to $N \cdot (10 \cdot \log_2 N + 13)$. The following summarizes the estimation of the reconstruction complexity:

- $4 \cdot S \cdot P \cdot N$ bytes of data must be read from the disk and $4 \cdot M$ bytes must be written back. If system memory is not large enough to store the projection data and the result, the computation has to be carried out in blocks. In this case the data is accessed in a non-contiguous way: only a few slices are read from each image file per iteration. Such access is significantly slower than sequential reading if present-day's magnetic hard drives are considered.
- If GPUs are used for processing, the source data must be transferred from the system to the GPU memory and the results must be transferred back to the system memory.
- The filtering step of the FBP algorithm needs $S \cdot P \cdot N \cdot (10 \cdot \log_2 N + 13)$ floating point operations to preprocess the source data.
- The back projection step needs $9 \cdot P \cdot M$ floating point operations to accomplish the reconstruction.

III. Optimizations

A. PyHST - High Speed Tomography in Python

PyHST is developed using two programming languages. The *Python* application is responsible for various management tasks and the *C* module is used to ensure good performance of computational intense parts. The *Python* application loads images, extracts slices for the current iteration, applies various corrections to the source data, and, finally, executes the *C*-code to reconstruct the image. The EDF (ESRF Data Format [25]) is the default image format of *PyHST*. Additionally, TIFF (Tagged Image File Format) and other formats implemented by the *Python-Imaging*, *ImageMagick* [26] and *VIPS* [27] libraries are supported. The filters are implemented using a plugin technique. The angles of the projections may be given either by specifying the inclination of the first projection and the angle between any two consecutive projections or by specifying the angle for all projections separately. The position of the rotational axis is specified in the reconstruction parameters and can be corrected for each slice individually.

B. New Architecture

As a first optimization step, the original application has been rewritten in a clean object-oriented way. The code was made thread-safe and both static and global variables were eliminated. The code was divided in several smaller objects: a reconstruction code, an error handler, a thread scheduler, a task manager, and a *Python* wrapper. The task manager is the main component. It performs initialization tasks and instructs the thread scheduler to start the processing of the slices. The initialization tasks include allocation of all temporary memory, configuration of the threading pool, and precomputation of constants shared among slices.

To enable simultaneous usage of GPU and CPU, the reconstruction code is implemented through an abstract interface. This interface defines initialization, cleanup, and process routines which are executed by the task manager at appropriate stages. The abstract reconstruction interface is implemented for the GPU and CPU. The original CPU version was slightly modified to implement the routines defined by the interface. The newly developed GPU version uses the *CUDA toolkit* to

shift all computations to the GPU. The details of the GPU implementation are described in section III-C.

The CPU code is optimized to use efficiently the available CPU cache and its structure is organized in a way helping the compilers with SIMD (Single Instruction Multiple Data) support to vectorize computations. The FFT transformations are computed using the *FFTW3* (Fastest Fourier Transformation in the West) API. This API is implemented by several libraries including the open-source *FFTW* and the highly optimized commercial *Intel MKL* (Math Kernel Library) [28], [29].

The redesigned *PyHST* may run in CPU, GPU, and hybrid modes. In the first two modes each thread in the pool is associated with a single CPU or GPU, respectively. The appropriate reconstruction module and consecutive number of CPU/GPU core are stored in the thread context. In the hybrid mode all computational resources of the system are used. In order to support multiple CPU and GPU devices, a simple thread scheduler is implemented. After all initialization tasks are carried out the scheduler runs all threads in the pool simultaneously. The threads request the next unprocessed slice from the scheduler and run the reconstruction using the associated module. The results are written directly to the output image file and the processing continues afterwards with the next slice. When all slices are processed, the threads are paused and the task manager returns control to the *Python* code which can load the next portion of slices from the files or terminate the execution if all slices are already processed. The only sequential points of execution are the request for new slices and writing out the result.

Multithreading and logging are implemented using *GLib* (Gnome Library [30]). The logging module of *GLib* is configured to pass all messages to the *Python-logging* facility and, hence, handle all log messages from both the *Python* and *C* parts of the application in a uniform way. Finally, the *Python* wrapper is used to encapsulate the calls to the task manager into the *Python/C interface*.

To simplify compilation on various platforms, a *CMake-based* build system is used to detect *PyHST* dependencies [31]. The *CMake* scripts check the availability and search for the installation paths of *Python*, *GLib* library, *FFTW3* and *Intel MKL* engines, *CUDA Toolkit*, etc. The scripts from the *FindCUDA* project are used to detect libraries and headers of *CUDA Toolkit and SDK* [32]. Depending on availability of the CUDA libraries, the CPU or GPU flavor of *PyHST* is built. However, the *CMake* configuration tool allows one to force building the CPU version even if all CUDA libraries are present. Using the *CMake* it is as well possible to select between *FFTW3* and *Intel MKL* libraries, request a single-threaded execution, and override the library paths.

C. GPU Implementation

Despite the availability of OpenCL, the current implementation is based on the CUDA architecture. The main reason is the maturity of this technology and a rich stack of available libraries. The OpenCL promises to become an industry standard for a variety of parallel architectures, independent of a particular vendor. However, it lacks the library support at the moment. NVIDIA does not provide an OpenCL version of their FFT and BLAS libraries. Third party libraries are pretty scarce or/and commercial. On the other hand, the OpenCL and CUDA technologies are architecturally similar. In order to translate GPU kernels from CUDA to OpenCL, only a few keywords need to be changed. The support code which is running on the CPU and scheduling GPU kernels needs only slightly more work. Using CUDA libraries, we were able to produce a first version of code in a very short time and, thanks to the similarity of architectures, we will rapidly port it to OpenCL when the technology becomes mature enough.

The preprocessed projection data is stored in 2D textures. The pixels of the reconstructed slice are divided into several groups at row boundaries. The groups are processed sequentially in a loop. Each pixel of the group is associated with a dedicated GPU thread and all pixels of the group are reconstructed by the CUDA kernel in parallel. Within the kernel a loop is executed over all projections. At each iteration, a projection bin corresponding to the reconstructed pixel is calculated and the texture fetching is performed. The texture engine is configured to carry out a linear interpolation while fetching the data. In this way both GPU multiprocessors and texture engines are used at the same time. This acceleration technique was first proposed in the beginning of the nineties for the *SGI RealityEngine* [23].

In order to preprocess the source data, a convolution with the configured filter is performed. The *cuFFT* library does not include any optimization for performing FFT transformations on data without imaginary part [33]. Another limitation of the *cuFFT* library is a strong dependency of its performance and accuracy on the size of the transformed vectors. In order to handle these limitations, the projection data is padded to the nearest power of two. To avoid the waste of complex computations, an approach to compute two real convolutions using a single complex one is utilized [34]. Two projections are interleaved in the GPU memory, transformed into the Fourier space using a single complex transform, multiplied with the filter, and back-transformed. This operation results in two filtered projections interleaved in the GPU memory. The projections are then copied into the texture memory and are later used in the back projection step of the algorithm. In order to improve the FFT performance even further, the complex projections are transformed in batched mode. However, not all transforms are batched together, but divided into several equally sized blocks. This enables one to exploit the notable feature of the two latest generations of NVIDIA cards to perform memory transfers between system and GPU memory in parallel with execution of computational kernels. While the current block is transferred, the previous one is processed using the specified Fourier filter.

The same approach is used while transferring the reconstructed image back to the system memory. The execution of the back projection kernel on a group of pixels is interleaved with memory transfers of already reconstructed pixels. In general, this results in almost completely hiding the transfer time within the computation time.

IV. Performance Evaluation

A. Hardware and Software Setup

Five systems available in our labs were compared as representatives of different families of platforms. The systems are not fully optimized but feature reasonable examples from each family. The performance of the CPU implementation is measured with a dual Xeon server.

System 1	Xeon Server (CPU)
Processor	Dual Xeon E5472
Motherboard	Supermicro X7DWE
Chipset	Intel 5400 chipset
PCI Express	PCIe 2.0, 36 lanes
Memory	24 GB DDR2-800 Memory
CPU	8 cores at 3 GHz
Price	$5,500

A simple low cost desktop system with a single graphic card represents our low-end GPU system.

System 2	Desktop (Single-Gpu)
Processor	Intel Core Duo E6300
Motherboard	Fujitsu-Siemens D3217-A
Chipset	Intel Q965 chipset
PCI Express	PCIe 1.1, 16 lanes, 1 GPU slot
Memory	4 GB DDR2-666 Memory
CPU	2 cores at 1.86 GHz
GPU	NVIDIA GTX 280
Price	$1,000

An advanced desktop solution of the latest generation is equipped with the *Asus Rampage III Extreme* motherboard which supports USB 3, the latest SATA 6 Gb/s interface, and up to four graphic cards which are connected using PCIe x8 if all four are installed and using PCIe x16 if only two cards are used. The system is currently equipped with two NVIDIA GeForce GTX295 adapters.

System 3	Advanced Desktop (Multi-GPU)
Processor	Intel Core i7 920
Motherboard	Asus Rampage III Extreme
Chipset	Intel X58 chipset
PCI Express	PCIe 2.0, 36 lanes, 4 GPU slots
Memory	6 GB DDR3-1333 Memory
CPU	4 cores at 2.66 GHz
GPU	2 x NVIDIA GTX295
Price	$1,800

A Tesla system from NVIDIA connected to a Xeon-based frontend server is used to evaluate performance of the professional series of GPU cards.

System 4	NVIDIA Tesla S1070 Server
Processor	Dual Intel Xeon E5472
Motherboard	Supermicro X7DWE
Chipset	Intel 5400 chipset
PCI Express	PCIe 2.0, 36 lanes
Memory	24 GB DDR2-800 Memory
CPU	8 cores at 3 GHz
GPU	4 x NVIDIA Tesla C1060
Price	$12,000

The high-end platform is a GPU server from Supermicro with 96 GB memory, SSD disks, and four GPU cards installed. The Supermicro motherboard is equipped with a dual Intel chipset with total number of 72 PCIe 2.0 lanes. Therefore, all four GPU adapters are running using a full x16 bandwidth.

System 5	Supermicro 7046GT GPU Server
Processor	Dual Intel Xeon E5540
Motherboard	Supermicro X8DTG-QF
Chipset	Dual Intel 5520 chipset
PCI Express	PCIe 2.0, 72 lanes, 4 GPU slots
Memory	96 GB DDR3-1066 Memory
CPU	8 cores at 2.53 GHz
GPU	2 x GTX480 + 2 x GTX295
Price	$8,000

All systems are running the 64 bit version of OpenSuSE 11.2 with the following software configuration:

- Linux Kernel 2.6.31.5
- GNU C Library 2.10.1
- GNU C Compiler 4.4
- Intel C Compiler 11.0.081
- Intel Math Kernel Library 10.2.1.017
- FFTW 3.3.2 (single-threaded SSE version)
- Gnome Library 2.22.1
- Python 2.6.2

B. Sample Data Set

In all tests reported below, 2000 projections were used to reconstruct a 3D image of a plastic holder with porous polyethylene grains. The image dimensions are $1691 * 1331 * 1311$ voxels and the projections have a size of $1776 * 1707$ pixels. According to section II-A, $600 * 10^9$ floating point operations are needed to filter the projection data and $53 * 10^{12}$ operations are necessary for the back projection. The amount of source data is about 24 GB and the resulting image has a size of 11 GB.

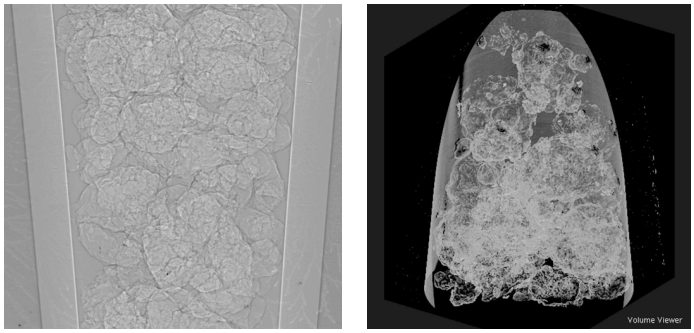

Fig. 3. The 2000 projections (left) were used to reconstruct a 3 gigavoxel image of porous polyethylene grains (right). The computation complexity of the reconstruction is about 54 TFlop. A total of 35 GB of data must be read and stored.

C. Compiler and FFT Library

The performance of an application is often dependent on the compiler and optimization flags used [35]. As a consequence,

to compare with the fastest possible CPU configuration, an optimal selection of *C* compiler and FFT library must be performed first. As it was described in earlier sections, the reconstruction process consists of filtering and back projection. The performance of the filtering step depends mainly on the speed of the FFT library. No external libraries are used in the back projection and the performance depends only on the compiler. The performance of popular compilers and FFT libraries available for the Linux platform is shown in Fig. 4. According to our results, the open-source *gcc-4.4* produces the best code for back projection, being slightly faster than the commercial *Intel C Compiler*. On the other hand *Intel Math Kernel Library* is significantly faster compared to the open-source alternatives. Therefore, in the following tests the *Intel Math Kernel Library* is used to perform filtering and all sources are compiled with *gcc-4.4* using *-O3 -march=nocona -mfpmath=sse* optimization flags.

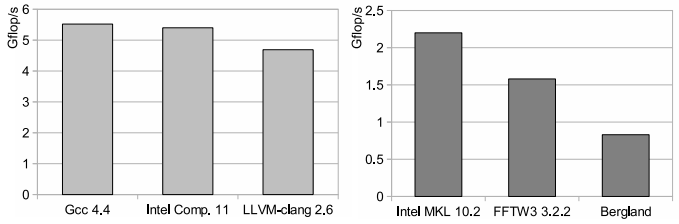

Fig. 4. Performance evaluation of *C* compilers (left) and FFT libraries (right). The test was run on a *Desktop* system and a single CPU version of *PyHST* was deployed. Only the performance of the back projection step was measured in the compiler benchmark and only the performance of the filtering step - in the FFT benchmark. For all compilers the SSE vectorization was switched on. With *gcc* and *clang* the following optimization flags were used: *-O3 -march=nocona -mfpmath=sse*. With the *Intel C Compiler* the employed optimization flags were: *-O3 -xS*. The *FFTW3* library was compiled with SSE support. The performance is measured in *GFlop/s* and larger values correspond to better results.

D. Reconstruction Performance

The performance of the five test systems is given in Fig. 5. If the GPU is used for image reconstruction even a cheap desktop is approximately 4 times faster than an expensive Xeon server. It takes only 40 seconds to reconstruct a 3 gigavoxel image from 2000 projections using the GPU server equipped with four graphic cards. At a price comparable with the Xeon server, the GPU server performs reconstruction 30 times faster.

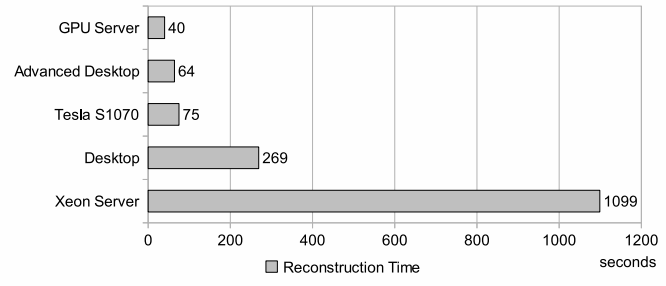

Fig. 5. Evaluation of the time needed to reconstruct the sample data set using different hardware platforms. The CPU-based reconstruction was performed on the Xeon server and only GPUs were used for all other platforms. Shorter times correspond to better results.

Our implementation scales well. According to Fig. 6, only 2.6% of the maximum possible performance is lost while the *Tesla* system is scaled up from one to four GPUs. The efficiency of the platforms in terms of *MFlop/s* per dollar ratio is shown in Fig. 7. It is easy to see that the desktop products are superior to the server based solutions using this metric. Furthermore, owing to the good scalability of algorithm implementation, the *Advanced Desktop* may be further enhanced by adding two more graphic cards.

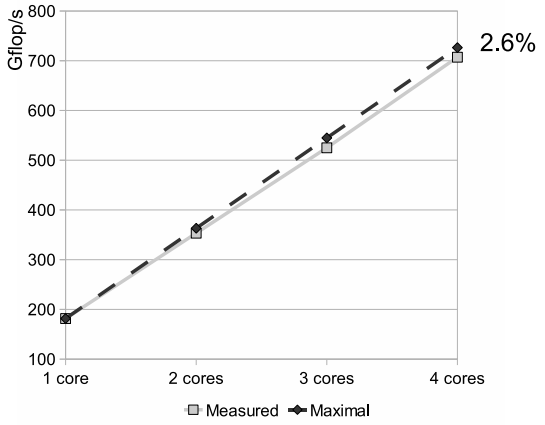

Fig. 6. Scalability evaluation of the GPU-based reconstruction. The test was executed on an NVIDIA Tesla S1070 system with 1 to 4 Tesla C1060 GPUs. The performance of the complete reconstruction process including the data transfer between host and GPU memory is measured. The dashed line indicates maximum possible performance (as if Flop rates of all GPUs would be just summed up). The evaluation shows that our implementation scales linearly with only 2.6% of performance loss in the case if all 4 GPUs are enabled.

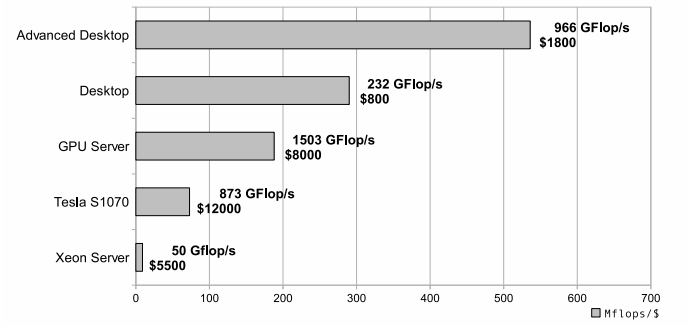

Fig. 7. *MFlop/s* per dollar efficiency of the tested hardware platforms for reconstruction using the back projection algorithm (filtering is omitted). Higher values correspond to better results. The actual performance in *GFlop/s* and the platform prices are shown in the chart.

E. Evaluation of NVIDIA Hardware

NVIDIA delivers multiple generations of graphic cards in consumer and professional versions. In this section the performance of the high-end graphic cards from the two latest generations is assessed. GeForce GTX295 includes two GT200 processors. GeForce GTX280 is its single processor counterpart. NVIDIA Tesla C1060 is a professional solution using the GT200 architecture. It has more memory than consumer products but at reduced clock rates. GeForce GTX480

is the current NVIDIA flagship product based on the latest GF100 (Fermi) architecture. All cards besides GTX295 have only a single processor. The following table summarizes the characteristics of the tested cards.

TABLE I
CHARACTERISTICS OF THE TESTED GRAPHICS ADAPTERS

GPU	GTX 480	GTX 295	GTX 280	C1060
Architecture	GF100	GT200	GT200	GT200
Processors	1	2	1	1
Parallel Threads	480	2 x 240	240	240
Clock Rate	1.4 GHz	1.25 GHz	1.3 GHz	1.3 GHz
Memory	1.5 GB	2 x 0.9 GB	1 GB	4 GB
Bandwidth	177 GB/s	2 x 112 GB/s	142 GB/s	102 GB/s
Texture Fill Rate	42 GT/s	2 x 46 GT/s	48 GT/s	48 GT/s
Power Consump.	250 W	289 W	236 W	188 W
Price	$500	$500	$250	$1200

As described above, the reconstruction process splits into three stages: data transfer, filtering using cuFFT, and back projection. In Fig. 8 the performance of the GPU cards for all these stages is investigated individually. In all tests the highest performance is provided by the dual-GPU GTX295. GTX480, a single-GPU card of the latest generation, is almost reaching the performance of GTX295 for the filtering step. The transfer bandwidth of GTX480 is also quite good compared to the single GPU cards of the older generation. However, as can be seen from TABLE I, the texture engine has not been significantly improved. As a consequence, the performance of the texture engine becomes the bottleneck for the back projection stage. The GTX480 is only slightly faster than the GTX280 and it is almost two times slower compared to the GTX295. The back projection step consumes 80% of the total reconstruction time and, as a result, for tomographic reconstructions the GF100 architecture is only slightly better than the older and cheaper GT200. Taking into account that the prices of GTX295 and GTX480 are approximately the same, it is significantly more efficient to use GTX295 cards. The professional series of Tesla cards have more memory on board but do not provide any performance benefits for tomographic reconstruction. The significantly cheaper consumer products are performing even a little better due to slightly faster clock rates.

Comparing the measured performance of GPU adapters to the peak values published by NVIDIA, a significant difference can be noted. Performing back projections the GTX295 achieves only 483 GFlop/s, which is about 25% of the theoretical peak performance (1788 GFlop/s). The filtering performance is even slower reaching only 70 GFlop/s, which constitutes 4% of the peak value. For comparison, actual and peak values are much closer in the CPU case. The *Intel Xeon E5472* exhibit a theoretical performance of 48 GFlop/s. For the back projection the actual performance is 25 GFlop/s, half of the nominal value. In the filtering code 9 GFlop/s, or 20% of the peak performance is reached. The utilization figures give an insight into the suitability of the algorithm for GPU architecture and code quality at the same time. The large difference between theoretical and real numbers is due to the extreme specialization of the graphic hardware. The optimal performance is only reached if a large amount of simple mathematical operations is executed simultaneously. Integer and double-precision arithmetic is significantly slower than single-precision. Access to global memory is time consuming and very strict access patterns must be followed for optimal performance. For some tasks the speed varies by more than a factor of 30 depending only on the memory access strategy [36]. In the case of the back projection, the Flop rate is bound by memory throughput as well. The performance of the texture engine is another limiting factor. However, even if the theoretical numbers are not reached, GPUs are still excellent number crunchers.

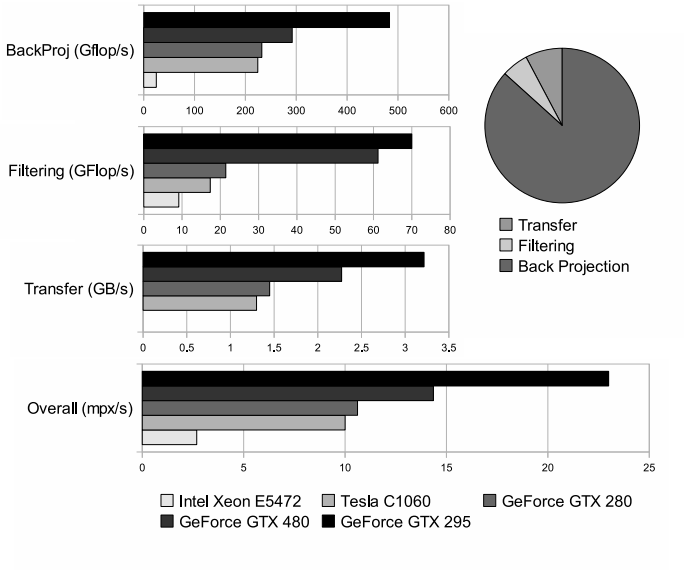

Fig. 8. The performance evaluation of NVIDIA hardware applied to tomographic reconstruction. The performance at all stages of reconstruction is measured independently. A pie-chart on the right shows the ratio between the times required for the stages. In order to measure precisely the impact of the stages, the data transfers and computations are not parallelized, but performed sequentially in this test. The overall performance is measured in megapixels per second and in all tests larger values correspond to better results.

V. I/O PERFORMANCE

Reconstruction is only one part of the task. The other part is data handling: the projections must be loaded into the system memory and the resulting 3D image must be stored. Besides, reading of projection data involves a sizable amount of random accesses which are considerably slow using commonly used magnetic hard drives. The I/O performance can be slightly improved if SSDs (Solid State Disk) are used as storage media. In contrast to magnetic hard drives, SSDs are based on microchips and do not include moving mechanical parts. This reduces random-access latencies significantly. The left part in Fig. 9 compares performance of traditional hard drives with the storage systems based on the SSD. A single SSD disk outperforms its magnetic counterpart by roughly 4 times. The performance can be increased few times more by combining several SSD drives in a RAID (Redundant Arrays of Inexpensive Disks).

The right part of the figure shows the ratio between the time spent in the computations and I/O operations whilst reconstructing the data on the GPU server. Using two *Intel*

X25-E SSD disks organized in a striping RAID-0, it was possible to significantly reduce the total time required for the data handling. Still, the I/O tasks take 4 times more time than reconstruction and remain the major performance bottleneck. To further reduce the I/O performance penalty, we are currently working on an implementation of a direct readout from the frame grabber. The GPU server is equipped with 96 GB of memory and is able to store both the source data and the resulting image directly in the system memory. On the long term, strategies to reduce the data stream such as data compression and frame reject are needed to further boost performance.

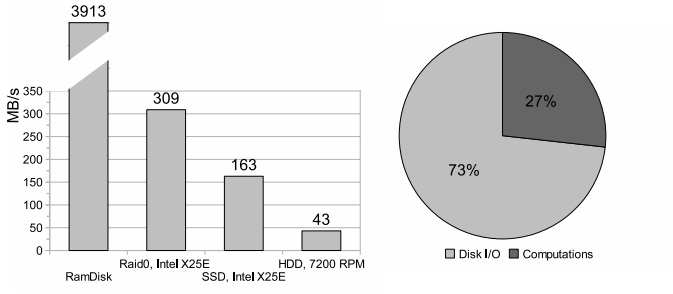

Fig. 9. Reconstruction of tomographic images requires reading and writing of large amounts of image data. The joint read/write throughput for different storage systems is depicted in the left chart. A *WDC5000AACS* SATA hard drive is compared with an *Intel X25-E* SSD disk, two such SSD disks organized in a striping RAID-0, and a virtual RAM disk. The *Ext4* file system is used in all cases. The right chart indicates the ratio of the time spent in computations and I/O respectively. The results were obtained using the GPU server with a storage system consisting of two *Intel X25-E SSD* drives assembled in RAID-0.

VI. Conclusion

Modern graphic cards provide over 1 TFlop/s of computational power and can be efficiently used to speed up scientific computations by more than one order of magnitude. The filtered back projection algorithm used for tomographic reconstruction can be implemented efficiently and with good scalability on the GPU architecture. Based on the source code of *PyHST* we have developed a GPU-based implementation of the algorithm which is able to exploit the computational resources of multiple GPU and CPU devices simultaneously. The performance evaluation confirms that a real-time assessment is possible with the current GPU architecture. Using a GPU server equipped with 4 graphic cards it takes only 40 seconds to reconstruct a 3 gigavoxel image from 2000 projections. This is approximately 30 times faster compared to the time needed to reconstruct the same image using an 8-core Xeon server which has the same price as the GPU server. Even a $1000 desktop equipped with a single GPU card outperforms the Xeon server by a factor of four.

We reduced the reconstruction time significantly. However, we are facing the challenge of rapidly loading multiple gigabytes of source data into the system memory. Using the fastest SSD disks assembled as RAID-0 we were able to perform all disk operations in approximately 2 minutes. However, this time is still inadequate compared to the computation time and currently we are working to read images directly into the system memory, bypassing the hard drive.

To build an optimal hardware setup, we have compared high-end *NVIDIA* cards from the current and last generations available on the market. Our evaluation has shown that the features provided by the professional *Tesla* series and the newly presented *Fermi* architecture do not improve the tomographic reconstruction significantly. Equipped with two *GT200* processors, the *GeForce GTX295* is the fastest adapter among the *NVIDIA* products. For better performance it is possible to stack up to 4 of such cards in a single system. *Supermicro* supplies the *7046GT* family of GPU servers, which include up to 4 GPU cards at full x16 speed, have two PCIe x4 extension slots for the frame grabber and a RAID adapter, and support up to 192 GB DDR3 memory to hold both the source projections and the resulting 3D image completely in the system memory. The cheaper alternative is a desktop system based on an *Asus Rampage III Extreme* motherboard. If equipped with two *GTX295* cards this system reaches 1 TFlop/s performance with a price below $2000. With a maximum of 48 GB memory supported, it is still possible to carry out most of the reconstructions directly in memory. The new SATA 3 (6 Gb/s) controller helps to reduce I/O time when used together with a RAID of SSD drives.

For the future, we also plan to extend this pipelined architecture from 2D slice-by-slice reconstruction to 3D volume reconstruction, as needed e.g. for synchrotron laminography [37].

References

[1] Y. Wang, F. de Carlo, D. C. Mancini, I. McNulty, B. Tieman, J. Bresnahan, J. I. I. Foster, P. Lange, G. von Laszewski, C. Kesselmann, M.-H. Su, and M. Thibaux, "A high-throughput X-ray microtomography system at the Advanced Photon Source," *Review of Scientific Instruments*, vol. 72, no. 4, pp. 2062–2068, 2001.

[2] F. Garcia-Moreno, A. Rack, L. Helfen, T. Baumbach, S. Zabler, N. Babcsan, J. Banhart, T. Martin, C. Ponchut, and M. DiMichiel, "Fast processes in liquid metal foams investigated by highspeed synchrotron X-ray micro-radioscopy," *Applied Physics Letters*, vol. 92, p. 134104, 2008.

[3] A. Rack, F. Garcia-Moreno, T. Baumbach, and J. Banhart, "Synchrotron-based radioscopy employing spatio-temporal micro-resolution for studying fast phenomena in liquid metal foams," *Journal of Synchrotron Radiation*, vol. 16, pp. 432–434, 2009.

[4] A. Rack, T. Weitkamp, S. B. Trabelsi, P. Modregger, A. Cecilia, T. dos Santos Rolo, T. Rack, D. Haas, R. Simon, R. Heldele, M. Schulz, B. Mayzel, A. N. Danilewsky, T. Waterstradt, W. Diete, H. Riesemeier, B. R. Müller, and T. Baumbach, "The micro-imaging station of the TopoTomo beamline at the ANKA synchrotron light source," *Nuclear Instruments and Methods in Physics Research Section B*, vol. 267, no. 11, pp. 1978–1988, 2009.

[5] C. Hintermueller, F. Marone, A. Isenegger, and M. Stampanoni, "Image processing pipeline for synchrotron radiation-based tomographic microscopy," *Journal of Synchrotron Radiation*, vol. 17, pp. 550–559, 2010.

[6] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[7] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[8] NVIDIA, "CUDA Zone." [Online]. Available: http://www.nvidia.com/object/cuda_home.html

[9] Dr. Dobb's, "Supercomputing for the masses." [Online]. Available: http://www.drdobbs.com/architecture-and-design/207200659

[10] ICL, "MAGMA - matrix algebra on GPU and multicore architectures." [Online]. Available: http://icl.cs.utk.edu/magma

[11] "GpuCV - GPU accelerated computer vision." [Online]. Available: https://picoforge.int-evry.fr/cgi-bin/twiki/view/Gpucv/Web/WebHome
[12] Khronos OpenCL Working Group, "The OpenCL specification," 2009. [Online]. Available: http://www.khronos.org/registry/cl/
[13] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*. IEEE press, 1988.
[14] J. Banhart, Ed., *Advanced Tomographic Methods in Materials Research and Engineering*. Oxford University Press, 2008.
[15] A. Hammersley and A. Mirone, "High speed tomography reference manual." [Online]. Available: http://www.esrf.eu/computing/scientific/HST/HST_REF/
[16] ASTRA research group of the University of Antwerp, "FASTRA II: the worlds most powerful desktop supercomputer." [Online]. Available: http://fastra2.ua.ac.be
[17] K. Mueller, "RapidCT - GPU accelerated tomography." [Online]. Available: http://www.cs.sunysb.edu/~mueller/research/rapidCT/index.htm
[18] V. Vlek, "Computation of filtered back projection on graphics cards," *WSEAS Transactions on Computers*, vol. 4, no. 9, p. 1216, 2005.
[19] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *Transactions on Nuclear Science*, no. 3, pp. 654–663, 2005.
[20] B. Amini, M. Björklund, R. Dror, and A. Nygren, "Tomographic reconstruction of SPECT data." [Online]. Available: http://www.owlnet.rice.edu/~elec539/Projects97/cult/report.html
[21] G. Zubal and G. Wisniewski, "Understanding fourier space and filter selection," *Journal of Nuclear Cardiology*, vol. 4, no. 3, 1997.
[22] L. Helfen, T. Baumbach, H. Stanzick, J. Banhart, A. Elmoutaouakkil, and P. Cloetens, "Viewing the early stage of metal foam formation by computed tomography using synchrotron radiation," *Adv. Eng. Mater.*, vol. 4, no. 10, pp. 808–813, 2002.
[23] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proc. of Symp. on Volume Visualization, Tysons Corner, Virginia, USA*, 1994, pp. 91–98.
[24] G. Bergland, "A fast fourier transform algorithm using base 8 iterations," pp. 275–279, 1968.
[25] ESRF, "Implementation of the EDF data format in the SAXS package." [Online]. Available: http://www.esrf.eu/UsersAndScience/Experiments/TBS/SciSoft/OurSoftware/SAXS/SaxsHeader
[26] "ImageMagick." [Online]. Available: http://www.imagemagick.org
[27] "VIPS image processing suite." [Online]. Available: http://www.vips.ecs.soton.ac.uk
[28] Intel, "Intel math kernel library (Intel MKL)." [Online]. Available: http://software.intel.com/en-us/intel-mkl/
[29] "FFTW." [Online]. Available: http://www.fftw.org/
[30] Gnome Foundation, "GLib reference manual." [Online]. Available: http://library.gnome.org/devel/glib/
[31] Kitware, "CMake - cross platform make." [Online]. Available: http://www.cmake.org
[32] J. Bigler, "FindCUDA." [Online]. Available: http://gforge.sci.utah.edu/gf/project/findcuda/
[33] NVIDIA, "CUFFT library," 2010. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/CUFFT_Library_3.0.pdf
[34] Engineering Productivity Tools Ltd, "The FFT demystified," 1999. [Online]. Available: http://www.engineeringproductivitytools.com/stuff/T0001/
[35] S. Chilingaryan, "The XMLBench project: Comparison of fast, multi-platform XML libraries," in *Database Systems for Advanced Applications: DASFAA 2009 International Workshops, Brisbane, Australia*, 2009, pp. 21–34.
[36] G. Ruetsch and P. Micikevicius, "Optimizing matrix transpose in CUDA," NVIDIA, Tech. Rep., 2009. [Online]. Available: http://www.cs.colostate.edu/~cs675/MatrixTranspose.pdf
[37] L. Helfen, T. Baumbach, P. Mikulík, D. Kiel, P. Pernot, P. Cloetens, and J. Baruchel, "High-resolution three-dimensional imaging of flat objects by synchrotron-radiation computed laminography," *Appl. Phys. Lett.*, vol. 86, no. 7, p. 071915, 2005.