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

Suren Chilingaryan, Andreas Kopmann, Alessandro Mirone, Tomy dos Santos Rolo

***Abstract*—Current imaging experiments at synchrotron beam lines often lack a real-time data assessment. X-ray imaging cameras installed at synchrotron facilities like ANKA provide millions of pixels, each with a resolution of 12 bits or more, and take up to several thousand frames per second. A given experiment can produce data sets of multiple gigabytes in a few seconds. Up to now the data is stored in local memory, transferred to mass storage, and then processed and analyzed off-line. The data quality and thus the success of the experiment, can, therefore, only be judged with a substantial delay, which makes an immediate monitoring of the results impossible. To optimize the usage of the micro-tomography beam-line at ANKA we have ported the reconstruction software to modern graphic adapters which offer an enormous amount of calculation power. We were able to reduce the reconstruction time from multiple hours to just a few minutes with a sample dataset of 20 GB. Using the new reconstruction software it is possible to provide a near real-time visualization and significantly reduce the time needed for the first evaluation of the reconstructed sample. The main paradigm of our approach is 100% utilization of all system resources. The compute intensive parts are offloaded to the GPU. While the GPU is reconstructing one slice, the CPUs are used to prepare the next one. A special attention is devoted to minimize data transfers between the host and GPU memory and to execute I/O operations in parallel with the computations. It could be shown that for our application not the computational part but the data transfers are now limiting the speed of the reconstruction. Several changes in the architecture of the DAQ system are proposed to overcome this second bottleneck. The article will introduce the system architecture, describe the hardware platform in details, and analyze performance gains during the first half year of operation.**

I. Introduction

DRIVEN by substantial developments in digital detector technology there is presently a tremendous progress going on in X-ray technology offering broad applications in the fields of medical diagnostics, homeland security, non-destructive testing, materials research and others. X-ray imaging permits spatially resolved visualization of the 2D and 3D structures in materials and organisms which is crucial for the understanding of their properties. Further it allows defect recognition in devices in a broad resolution range from the macro down to the nano-scale. Additional resolution in the time domain gives insight in the temporal structure evolution and thus access to dynamics of processes allowing to understand functionality of devices and organisms and to optimize technological processes [1], [2].

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

A. Mirone is with European Synchrotron Radiation Facility, Grenoble, France

T. dos Santos Rolo is with Institute for Synchrotron Radiation, Karlsruhe Institute of Technology, Karlsruhe, Germany

In recent years, synchrotron tomography has seen a tremendous decrease of total acquisition time [3]. Based on available photon flux densities at modern synchrotron sources ultra-fast X-ray imaging could enable investigation of the dynamics of technological and biological processes with a time scale down to the milliseconds range in the 3D. Using modern CMOS based detectors it is possible to reach up to several thousand frames per second. For example, frame rates of 5000 images per second were achieved using a filtered white beam from ESRF's ID19 wiggler source [4]. Using a larger effective pixel size the frame rates of 40000 images per second were reported [5]. As a result of improved image acquisition, a given experiment can produce data sets of multiple gigabytes in a few seconds. There is a major challenge to process these data in reasonable amount of time facilitating on-line reconstruction and fast feedback.

Two main approaches are used at the moment to handle the huge data-sets produced at the tomographic setups.

- At *TopoTomo* beam-line of ANKA (synchrotron facility at KIT [5]) up to now the data was stored in local memory, transferred to mass storage, and then processed and analyzed off-line. The data quality and thus the success of the experiment can only be judged with a substantial delay, which makes an immediate monitoring of the results impossible.
- Second alternative is supercomputer-based processing, expensive in terms of money and power consumption. For example, a pipelined data acquisition system combining a fast detector system, high speed data networks, and massively parallel computers was employed at synchrotron at Aragonne National Laboratory to acquire and reconstruct a full tomogram in tens of minutes [3].

Instead we decided to use computation power of modern graphic adapters provided by NVIDIA and AMD. Including hundreds of simple processors used to transform vertexes in 3D space they offer a way to speed up the process of more than one order of magnitude at low cost and with good scalability. A peak performance of fastest graphic cards exceeds one teraflop (for single precision numbers). Top gaming desktops include up to four of such cards and, thus, producing about 5 teraflops of computational power. Compared to 100 gigaflops provided by commonly used servers, this gives a potential speedup of 50 times [6], [7].

To provide this enormous computational power to developers NVIDIA released a CUDA (Common Unified Device Architecture [8]) toolkit. It extends C language with few syntactical constructs used for GPU programming. CUDA exploits data parallelism: basically the architecture allows to execute a sequence of mathematical operations, so called kernels, simultaneously over a multi-dimensional set of data. API includes functions to transfer the data from the system memory to GPU memory, execute kernels to process these data, and transfer the results back to the system memory. For developer convenience NVIDIA and community have released a number of libraries implementing many standard algorithms. NVIDIA 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 community implementations of LAPACK (Linear Algebra PACKage) and quantity of computer vision algorithms [9], [10], [11].

In addition to vendor-dependent programming toolkits, like CUDA, Khronos Group have 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 to execute developed application on both AMD and NVIDIA GPUs as well as on general-purpose multi-core/multiprocessor CPUs.

FBP (Filtered Back Projection) is a standard method to reconstruct 3D image of object from multiplicity of tomographic projections [13]. PyHST (High Speed Tomography in Python [14]) is a fast implementation of filtered back-projection developed at ESRF (European Synchrotron Radiation Facility) and used currently at synchrotrons at ESRF and KIT [5]. While it uses heavily optimized *C*-code for performance intensive parts of algorithm, the processing of typical data set with nowadays Xeon servers is still requiring about one hour of computations. In order to increase speed of processing and achieve a near real-time performance we accelerated reconstruction by offloading most of the computations to GPU with the help of CUDA toolkit. The optimized version of PyHST is able to reconstruct a typical 3D image (3 gigavoxel image is 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 8-core Xeon server used before.

There are a few other research projects aiming to GPU-assisted tomographic reconstruction. The group from university of Antwerp has assembled a GPU server with 7 NVIDIA GTX 295 cards reaching 12 TFlops of theoretical peak performance. The presented benchmark show 35 times speedup compared with Intel Core i7-940 [15]. RapidCT is another project developing GPU-assisted tomographic software. The presented results indicating 20 times performance increase [16]. As well there are earlier implementations of filtered back projection developed using shader language [17], [18]. Unfortunately, no project is distributing the developed software. On the other hand PyHST with our optimizations is available under *GNU Public License* from our web site [19].

The rest of article is organized as follows. Second section describes the filtered back-projection algorithm and provides an estimation of computation complexity. In third section the details about *PyHST* and our optimizations are presented. The performance evaluation, a discussion of I/O bottleneck, and a review of available hardware platforms are provided in fifth section. Finally, conclusion summarizes presented information.

II. Imaging at Synchrotron Light Sources

Fundamentally, the imaging at synchrotron light sources is working as follows. The sample is placed at horizontal platform and evenly rotated in front of a pixel detector. The parallel X-rays are penetrating the sample and the pixel detector registers series of parallel 2D projections of the sample density at different angles. These projections along with the coordinates of the rotation axis and the angle between slices are the input data for PyHST. To reconstruct a 3D image from projection PyHST employs a FBP algorithm. All projections are replicated in 3D space across the X-rays direction and integrated, see Fig. 1. Since the sample is rotating round the strictly vertical axis, the sample can be seen as a sequence of horizontal slices which can be reconstructed separately. If the center of rotation is located at a center of coordinate system then to find a value of point with coordinates (x, y) in slice z, it is necessary to compute the following sum over all projections: $\sum_{p=1}^{P} I_p(xcos(p\alpha) - ysin(p\alpha), z)$ where P is a number of projections, α is an angle between projections, and I_p is a p-th projection. Since the projections are digital images of finite resolutions, the linear interpolation is performed to get density value at position $xcos(p\alpha) - ysin(p\alpha)$ [13], [20].

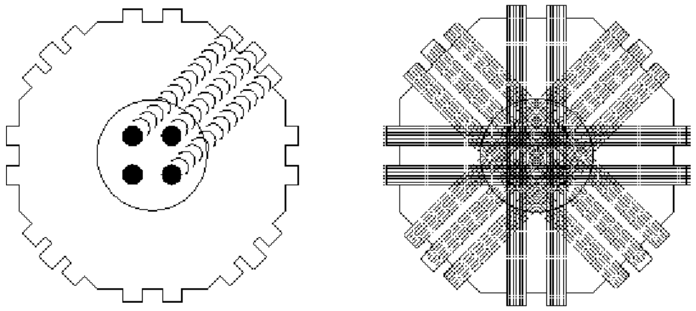
Fig. 1. Image reconstruction using back-projection algorithm. The sample is evenly rotating and pixel detectors registers series of parallel projections at different angles (left). All registered projections are replicated across direction of incidence and integrated (right).

Unfortunately, described approach is yielding to blurred results. As it can be seen from Fig. 1 besides the circular objects in the center the reconstructed image contains faint traces of the prolonged projections. This effect diminishes with the number of projections, but in the standard setups is high enough to blur details at the boundary of changing density. To compensate the blurring effect the raw projection data is multiplied in the Fourier domain with the ramp filter before being back-projected. However, this approach intensifies noise and in some case can introduce aliasing effects. To reduce these effects other types of filters are used instead of the ramp filter in specific cases. Fig. 2 illustrates difference between unfiltered back projection and back projection using the ramp filter [13], [21].

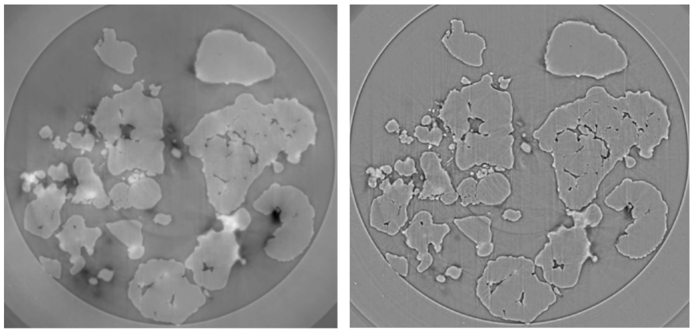

Fig. 2. A reconstructed slice of a conical plastic holder with porose polyethylene grains. A ramp filter was used to reconstruct right image and no filtering was performed for the left.

A. Algorithm Complexity

The source data includes S slices consisting of P projections and N bins in each projection. The reconstructed 3D image should be composed of M voxels totally. Both the source and the result data is stored uncompressed in a single-precision floating point format (32 bit). If linear interpolation is used the total number of operations required to back-project the image is equal to $9 * P * M$ [22]. To filter a single projection a convolution should be computed for each vertical slice individually. Therefore, the complexity of the filtering step consists from the computation of direct and inverse FFT transforms of projection slices and a multiplication with the filter in a Fourier domain. According to Bergland if the number of elements in transformed vector (N) is equal to power of 2, the exact number of floating point operations required to compute FFT is $N(alog_2N + 6)$, where a is between 4.03125 and 5 depending on a radix of algorithm [23]. Further, for the simplicity reasons $N(5log_2N + 6)$ is used to estimate computation complexity of FFT. And, then, the total number of operations required to filter a single slice is equal to $N(10log_2N + 13)$.

The following list summarizes the complexity estimation:

- $4 * S * P * N$ bytes of data should be read from the disk and $4 * M$ bytes should be written back. If the amount of system memory is not enough to completely store the projection data and result, the data should be processed slice by slice. In this case the data is accessed in non-contiguous way: only a few slices are read from each image file per iteration. Described access pattern is significantly slower than sequential read if nowadays magnetic hard drives are considered.
- If the GPUs are used for processing, the source data should be transferred from the system memory to GPU memory and the results should be transferred back to the system memory.
- A filtering step of FBP algorithm needs $S * P * N * (10log_2N + 13)$ floating point operations to preprocess the source data.
- The main step needs $9 * P * M$ floating point operations to accomplish the reconstruction.

III. Optimizations

A. PyHST

PyHST is developed using two programming languages: *Python* part is responsible for various management tasks and *C* code is used for performance critical parts. The *Python* application loads images, extracts the slices for current iteration, applies various corrections to the loaded data, prepares the data for filtering step, and, finally, executes *C*-code to reconstruct the image. Current version of *PyHST* supports only an *EDF* (ESRF Data Format [24]) file format developed by ESRF. However, the converters from *TIFF* (and other image formats supported by *ImageMagick* [25]) are provided and will be included in the next version of *PyHST*. The filters are implemented using plugins. The plugin accepts a size of a filter core and produces an array of appropriate size which would be multiplied with input data in the Fourier domain. The angles of projections may be given in two ways: by specifying the declination of the first projection and the angle between any two consequential projections or by specifying the declination angle for all projections separately. The position of rotational axis is specified in the reconstruction parameters and can be corrected for each slice individually.

B. New Architecture

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

To allow simultaneous usage of GPU and CPU the reconstructors are implemented through an abstract interface. The interface defines initialization, cleanup, and process routines which are executed by the task manager at appropriate stages. Two implementations of the abstract interface are available now: CPU and GPU. The original CPU version is slightly modified to implement the routines defined by the interface. The newly developed GPU version uses *CUDA toolkit* to offload all computations to the GPU. It is described in more details in the next subsection.

The CPU code is optimized to efficiently use available L2 cache and its structure organized in a way helping the compilers with SIMD support to vectorize computations. The FFT transformations are computed using *FFTW3* (Fastest Fourier Transformation in the West) API. This API is implemented by several libraries including open-source *FFTW* and highly optimized commercial *Intel MKL* (Math Kernel Library) [26], [27].

The *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 correspondingly. 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 simultaneously runs all threads in the pool. The threads are requesting the next unprocessed slice from the scheduler and run the reconstruction using the associated module. Then, the results are written directly to the resulting image file and the next slice is requested. When all slices are processed, the threads are paused and the task manager returns control to python code which could load next portion of slices from the files or terminate execution if all slices are already processed. The only sequential points of execution are the request of new slices and the result write out.

The multithreading and logging are implemented using *GLib* (Gnome Library [28]). The logging module of *GLib* is configured to pass all messages to *python-logging* and, hence, to perform handling of all log messages from both *Python* and *C* parts of 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 build system* is used to detect *PyHST* dependencies [29]. The *CMake* scripts are checking the availability and find installation paths of the *Python*, the required *Python* modules, the *GLib* library, *FFTW3* and *Intel MKL* engines, the *CUDA Toolkit*, etc. The *FindCUDA* project is used to detect libraries and headers of *CUDA Toolkit and SDK* [30]. Depending on availability of CUDA libraries, the CPU or GPU flavor of *PyHST* is built. However, a *CMake* configuration tool allows to force build of CPU version even if all CUDA libraries are present. As well using the configuration tool it is possible to select between *FFTW3* and *Intel MKL* libraries, request a single-threaded execution, and override the library paths.

C. GPU Implementation

Despite availability of OpenCL, the current implementation is based on CUDA architecture. The main reason is the maturity of technology and a rich stack of available libraries. While OpenCL promises stable industry standard and vendor independence, currently it strongly misses the software support. NVIDIA does not provide OpenCL version of their FFT and BLAS libraries. The third party libraries are pretty scarce or/and commercial. On the over hand the OpenCL and CUDA technologies are architecturally similar. In order to translate GPU kernels from CUDA to OpenCL only a few keywords should be changed. The support code which is running on CPU and scheduling GPU kernels needs only a slightly more work. Therefore, using CUDA libraries we were able to produce a first version in a very short time and, thanks to the similarity of architectures, we will rapidly port it to OpenCL when the technology is mature enough.

The back projection code is extremely simple. The projection data is stored in 2D textures. The pixels of reconstructed slice are divided into the several groups at row boundaries. The groups are processed sequentially in a loop. All pixels of a single group are reconstructed by the CUDA kernel in parallel. Within the kernel a loop over all projections is executed. At each iteration, a projection bin corresponding to the reconstructed pixel is calculated and texture fetch is performed. The texture engine is configured to perform the linear interpolation while fetching the data. In this way both GPU multiprocessors and texture engines are used in the same time. The technique was proposed for acceleration using *SGI RealityEngine* as early as 1994 by B. Cabral [22].

To perform the filtering of source data a convolution with the configured filter is executed. Unfortunately, the *cuFFT* library does not include any optimization for performing FFT transforms on a pure real data [31]. Another limitation of *cuFFT* library is high dependency of performance and accuracy on a transform size. To handle last problem the projection data is padded to the nearest power of two. To avoid wastage of the complex computation an approach to compute two real convolutions using a single complex one is utilized [32]. Two projections are interleaved in GPU memory, transformed into the Fourier space using a single complex transform, multiplied with the filter, and transformed back. This operation results in two filtered projections interleaved in GPU memory. The projections are, then, copied into the texture memory and are later used in the back-projection step of algorithm. In order to further increase FFT performance, the described complex projections are transformed using *cuFFT* batched transform. However, not all transforms are batched together, but divided in several equally sized blocks. This allows exploiting a feature of few latest generations of NVIDIA cards to perform memory transfers between system and GPU memory in parallel with execution of computation kernels. While the current block is transferred, the previous one is filtered using the specified Fourier filter.

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

IV. Performance Evaluation

A. Hardware and Software Setup

Four different systems were used to evaluate efficiency of a GPU version compared to the CPU solution. A simple low cost desktop system with a single GPU adapter.

System	Desktop
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$

Advanced desktop of the latest generation. *Asus Rampage III Extreme* motherboard supports USB 3, 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 actually equipped with two NVIDIA GTX295 adapters.

System	Advanced Desktop
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 GPU server from Supermicro with lots of memory, SSD disks, and four GPU cards installed. The Supermicro motherboard is equipped with dual Intel chipset with total number of 72 PCIe 2.0 lanes. Therefore, all four GPU adapters are running using full x16 bandwidth.

System	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$

And finally an NVIDIA Tesla system with Xeon-based frontend server.

System	NVIDIA Tesla S1070
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$

To measure a CPU performance, a dual Xeon server was used.

System	Xeon Server
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$

All systems were running 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 below 2000 projections was used to reconstruct 3D image of plastic holder with porose polyethylene grains. The image dimensions are $1691 * 1331 * 1311$ and the projections had size of $1776*1707$ pixels. According to computations in the section II-A $600 * 10^9$ floating point operations are needed to filter the projection data and $53 * 10^{12}$ operations are needed for the back-projection. The size of source data is about 24 GB and the resulting image is 11 GB.

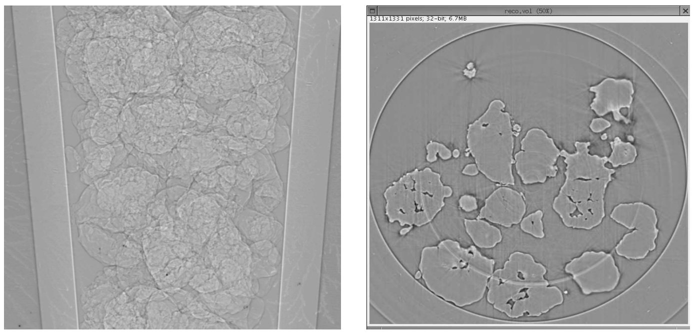

Fig. 3. The 2000 projections (example is shown on the left side) were used to reconstruct a 3 gigavoxel image of a porose polyethylene grains (one slice is shown on the right side). The computation complexity of the reconstruction is about 54 TFlop and 35 GB of data should be read and stored.

C. Compiler and FFT Library

The application performance is often dependent on the compiler and optimization flags used [33]. Therefore, to make a fair comparison between CPU and GPU versions an optimal selection of *C* compiler and FFT library should be performed first. As it was described in earlier sections the reconstruction process consists of filtering and back-projection steps. The performance of the filtering step depends mainly on the speed of FFT library. No external libraries are used in the back-projection and the performance depends only on the compiler. The Fig. 4 evaluates performance of the most popular compilers and FFT libraries available for Linux platform. According to the 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 with 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*. The following optimization flags are used: *-O3 -march=nocona -mfpmath=sse*.

D. Performance Evaluation

Fig. 5 compares performance of all described systems. As can be seen from the chart if the GPU is used for image

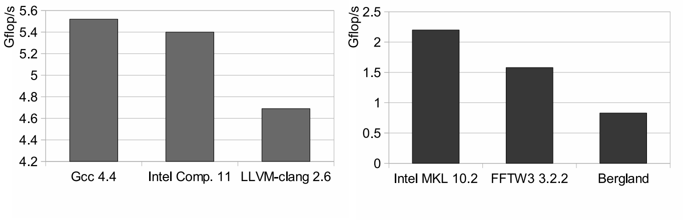

Fig. 4. Performance evaluation of *C* compilers (left) and FFT libraries (right). The test was run on a *Desktop* system and a single CPU version of *PyHST* was executed. The back-projection performance was measured in the compiler benchmark and the filtering performance - in the FFT benchmark. For all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

reconstruction, even a cheap desktop is approximately 4 times faster than expansive 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. Having a price comparable with the Xeon server, the GPU server performs reconstruction 30 times faster. Our implementation scales well. According to Fig. 6 only 2.5% of maximum possible performance is lost while Tesla system is scaled from one to four GPUs. Fig. 7 evaluates *MFlop/s* per dollar efficiency of the platforms. It is easy to see that the desktop products are extremely good using this metric. It should be noted that *Advanced Desktop* may be simply enhanced by adding two more graphic cards.

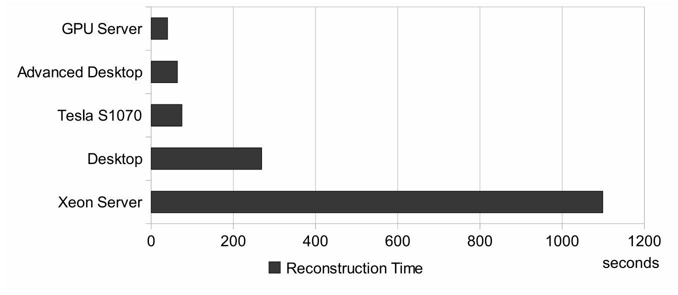

Fig. 5. The chart evaluates the time needed to reconstruct the sample data set using different hardware platforms. The CPU-based reconstruction was performed for the Xeon server and only GPUs were used for all other platforms. The smaller times correspond to the better results.

E. I/O Performance

Unfortunately, the reconstruction is only part of the task. First, the projections should be loaded into the memory and, in the end, the produced image should be written to the storage. The right part of Fig. 8 shows the ratio between time spent in the computations and I/O operations while reconstructing the data using GPU server. The I/O takes almost 10 times more time than reconstruction with the current version of *PyHST*. According to the left part of the figure usage of SSD (Solid State Disk) media, especially several SSD drives assembled in a raid array, could slightly relax the problem, but still I/O remains a major bottleneck.

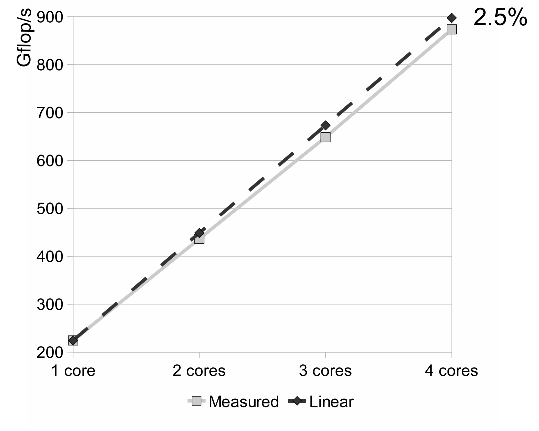

Fig. 6. The chart evaluates scalability of a GPU-based implementation of the back-projection algorithm. The test was executed on NVIDIA Tesla S1070 system with 1 to 4 Tesla C1060 GPUs enabled. Deviation from the linear scalability is shown.

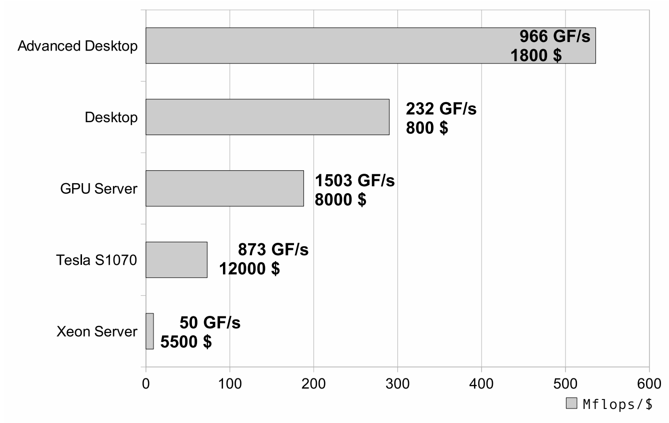

Fig. 7. The chart evaluates *MFlop/s* per dollar efficiency of the tested hardware platforms for reconstruction using back-projection algorithm (filtering is omitted). The higher values are corresponding to the better results. The actual performance in *GFlop/s* and the platform price are shown on the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

While 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, the usage of *RamDisk* does not significantly improve situation. In order to address the problem, we are at the moment working on an implementation of a direct readout from the frame grabbers. Besides, the FPGA-based frame compression and reject probably could be used to reduce the data rate.

F. Evaluation of NVIDIA Hardware

NVIDIA delivers multiple generations of graphic cards in the consumer and professional versions. In this section a performance of top graphic cards from the two latest generations is evaluated. NVIDIA GTX295 includes two GT200 processors. NVIDIA GTX280 is its single processor counterpart. NVIDIA Tesla C1060 is professional solution using GT200 architecture. It has more memory than consumer

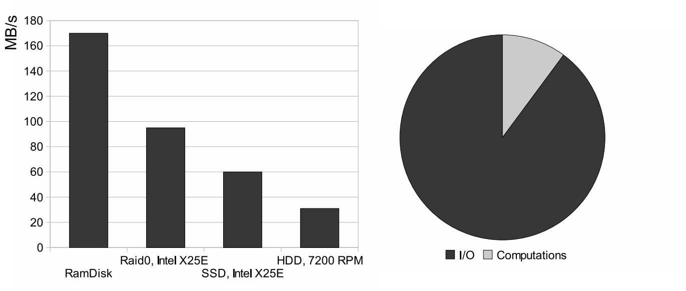

Fig. 8. Reconstruction of the tomographic images requires reading and writing of big amounts of image data. Using the systems with limited amount of memory, the reading of projection data involves significant amount of random accesses. The joint read/write throughput for different storage systems are measured on the right chart. A *WDC5000AACS* SATA hard drive is compared with *Intel X25-E* fast SSD disk and two such SSD disks organized in a striping raid. The results for virtual disk stored completely in the system memory are presented as well. The directory structure is organized using *Ext3* file system in all cases. The right chart indicates the ratio of time spent in computations and I/O. The result is obtained using a GPU server with storage system consisting of two *Intel X25-E SSD* drives assembled in Raid-0.

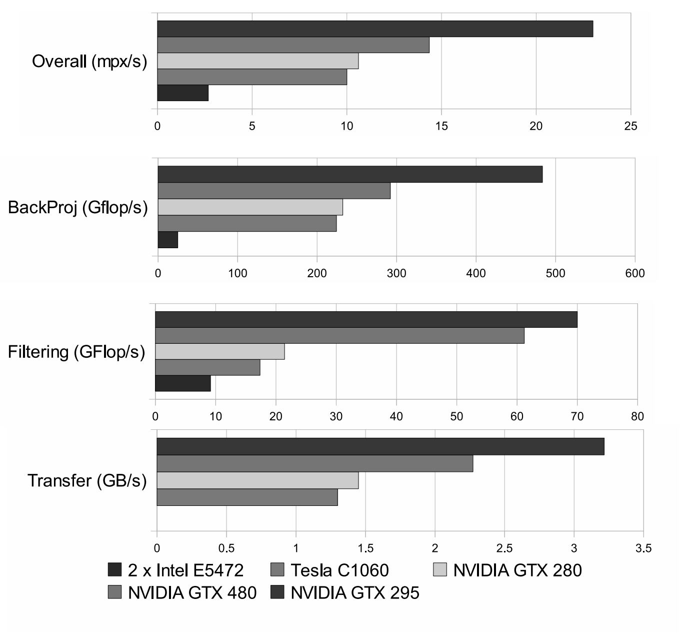

Fig. 9. The performance evaluation of NVIDIA hardware applied for tomographic reconstruction. The performance of all three stages of processing is measured independently. The overall performance is measured in megapixels per second, the back-projection and filtering in *GFlop/s* per second, and transfer in *GB* per second. The transfer bandwidth in both directions is measured jointly. Bigger results are corresponding to better results.

market products but reduced clock rates. GTX480 is a current NVIDIA flagship based on the latest GF100 architecture. All cards besides GTX295 have only a single processor.

As it was described above, the reconstruction process can be split in three stages: data transfer, filtering using cuFFT, and back-projection. Fig. 9 evaluates performance of GPU cards for all these stages individually. As can be seen from the chart in all tests the top performance has GTX295, a dual-GPU card. GTX480, a single-GPU card of the latest generation, is almost reaching performance of GTX295 for the filtering step. The transfer bandwidth of GTX480 is also quite good compared to the single-GPU cards of older generation. However, the back-projection performance is only insignificantly faster and almost two times slower if compared with GTX295. Unfortunately, according to Fig. 10 the back-projection step consumes 80% of reconstruction time and, as a result, for tomographic reconstruction 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 same, it is significantly more efficient to use GTX295 cards.

The professional series of Tesla cards have more memory on board but does not provide any performance benefits for tomographic reconstruction. The significantly cheaper consumer market products are performing even a bit better due a slightly faster clock rates.

V. Conclusion

The modern graphic cards provides enormous amount of computational power and can be efficiently used to speedup scientific computations at orders of magnitude. The FBP (Filtered Back Projection) algorithm used for tomographic reconstruction is extremely well fitting to the GPU architecture. Basing on the source of *PyHST* we have developed a GPU-based implementation of an FBP algorithm. The remodeled architecture of *PyHST* allowed us to exploit the computational resources of multiple GPU and CPU devices simultaneously. The performance evaluation confirms the superiority of GPU version. Using a GPU server equipped with 4 graphic cards it takes only 40 second to reconstruct a 3 gigavoxel picture from 2000 projections. This is approximately 30 times faster compared to the time needed to reconstruct the same image using 8-core Xeon server costing the same money. Even under-a-$1000-desktop equipped with a single GPU card outperforms the Xeon server by approximately 4 times.

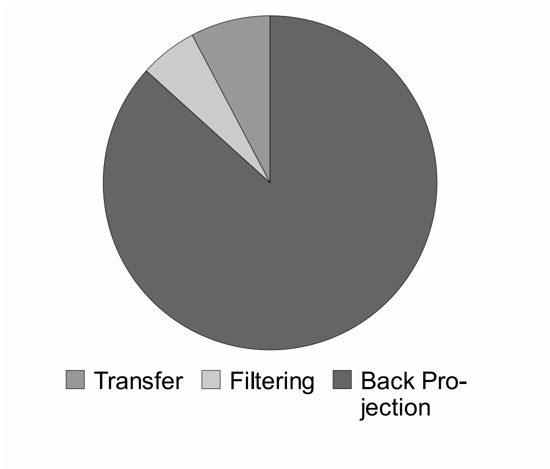

Fig. 10. The ratio between times required for different stages of the reconstruction process.

Significantly reducing the reconstruction time we, however, are facing the challenge to rapidly load multiple gigabytes of the source data into the system memory. With a standard 7200 RPM hard drive it takes only 40 seconds to reconstruct the image, but whole 15 minutes for the disk I/O. We were able to slightly improve situation by using multiple SSD disks assembled in Raid-0. The I/O time was reduced to 5 minutes only. However, this time is still inadequate to computation time and at the moment we are working to load images directly from the frame grabber into the system memory bypassing the hard drive.

To build optimal hardware setup we have compared top NVIDIA cards available on the market. Our evaluation has indicated that for tomographic reconstruction neither the performance of the new Fermi architecture nor the performance of professional Tesla series are significantly differing from the performance of GT200 series of consumer products. GTX295 card equipped with two GT200 processors are two times faster than any other NVIDIA product. For the best performance, it is possible to stack up to 4 such cards in a single system. Supermicro supplies 7046GT family of GPU servers. The dual-chipset motherboard from Supermicro has 72 PCIe (gen2) lanes and supports up to 4 GPU cards at full x16 speed. Additionally, the board has two PCIe x4 slots for the frame grabber and raid adapter. It supports up to 192 GB of DDR3 memory in 12 slots allowing storage of both the source projections and the resulting 3D image completely in the system memory. The cheaper alternative is a desktop system based on *Asus Rampage III Extreme* motherboard. With the price under \$2000 it is possible to reach performance of 1 TFlop/s for the back-projection. The board has 4 GPU slots and supports 4 graphic cards at x8 speed or 2 at x16. With 48 GB of maximally supported memory it is still possible to process most of the expected reconstructions directly in the memory. The new SATA 3 (6 Gb/s) controller embedded in ASUS board helps to significantly reduce I/O time if used together with a striped raid of SSD drives.

Both CPU and GPU versions of *PyHST* are licensed under GPL and available online.

References

[1] J. H. Kinney, S. R. Stock, M. C. Nichols, U. Bonse, T. Breunig, R. Saroyan, R. Nusshardt, Q. C. Johnson, F. Busch, and S. D. Antolovich, "Nondestructive investigation of damage in composites using x-ray tomographic microscopy (xtm)," *Journal of Materials Research*, vol. 5, no. 5, pp. 1123–1129, 1990.

[2] T. Weitkamp, A. Diaz, C. David *et al.*, "X-ray phase imaging with a grating interferometer," *Opticts express*, vol. 13, no. 16, pp. 6296–6304, 2005.

[3] 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.

[4] F. Garca-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, pp. 134 104–1–134 104–3, 2008.

[5] 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. Mller, 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.

[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. Dobbś, "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] A. Hammersley and A. Mirone, "High speed tomography reference manual." [Online]. Available: http://www.esrf.eu/computing/scientific/HST/HST_REF/

[15] ASTRA research group of the University of Antwerp, "Fastra ii: the worlds most powerful desktop supercomputer." [Online]. Available: http://fastra2.ua.ac.be

[16] K. Mueller, "Rapidct - gpu accelerated tomography." [Online]. Available: http://www.cs.sunysb.edu/~mueller/research/rapidCT/index.htm

[17] V. Vlek, "Computation of filtered back projection on graphics cards," *WSEAS Transactions on Computers*, vol. 4, no. 9, p. 1216, 2005.

[18] 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.

[19] S. Chilingaryan, "Gpu-optimized version of pyhst: Svn and bug tracker." [Online]. Available: http://dside.dyndns.org/PyHST

[20] B. Amini, M. Bjrklund, 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] 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.

[23] G. Bergland, "A fast fourier transform algorithm using base 8 iterations," pp. 275–279, 1968.

[24] ESRF, "Implementation of the edf data format in the saxs package." [Online]. Available: http://www.esrf.eu/UsersAndScience/Experiments/TBS/SciSoft/OurSoftware/SAXS/SaxsHeader

[25] "Imagemagick." [Online]. Available: http://www.imagemagick.org

[26] Intel, "Intel math kernel library (intel mkl)." [Online]. Available: http://software.intel.com/en-us/intel-mkl/

[27] "Fftw." [Online]. Available: http://www.fftw.org/

[28] Gnome Foundation, "Glib reference manual." [Online]. Available: http://library.gnome.org/devel/glib/

[29] Kitware, "Cmake - cross platform make." [Online]. Available: http://www.cmake.org

[30] J. Bigler, "Findcuda." [Online]. Available: http://gforge.sci.utah.edu/gf/project/findcuda/

[31] NVIDIA, "Cufft library," 2010. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/CUFFT_Library_3.0.pdf

[32] Engineering Productivity Tools Ltd, "The fft demystified," 1999. [Online]. Available: http://www.engineeringproductivitytools.com/stuff/T0001/

[33] 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.