Rapid Data Processing for Ultrafast X-Ray Computed Tomography Using Scalable and Modular CUDA based Pipelines

Frust, T.; Wagner, M.; Stephan, J.; Juckeland, G.; Bieberle, A.;

Originally published:

June 2017

Computer Physics Communications 219(2017), 353-360

DOI: https://doi.org/10.1016/j.cpc.2017.05.025

Rapid Data Processing for Ultrafast X-Ray Computed Tomography Using Scalable and Modular CUDA based Pipelines

Tobias Frust[a,*], Michael Wagner[b], Jan Stephan[a], Guido Juckeland[a], André Bieberle[a]

[a] *Helmholtz-Zentrum Dresden-Rossendorf, Bautzner Landstr. 400, 01328 Dresden, Germany*
[b] *Technische Universität Dresden, AREVA Endowed Chair of Imaging Techniques in Energy and Process Engineering, 01062 Dresden, Germany*

Abstract

Ultrafast X-ray tomography is an advanced imaging technique for the study of dynamic processes basing on the principles of electron beam scanning. A typical application case for this technique is e.g. the study of multiphase flows, that is, flows of mixtures of substances such as gas-liquid flows in pipelines or chemical reactors. At Helmholtz-Zentrum Dresden-Rossendorf (HZDR) a number of such tomography scanners are operated. Currently, there are two main points limiting their application in some fields. First, after each CT scan sequence the data of the radiation detector must be downloaded from the scanner to a data processing machine. Second, the current data processing is comparably time-consuming compared to the CT scan sequence interval. To enable online observations or use this technique to control actuators in real-time, a modular and scalable data processing tool has been developed, consisting of user-definable stages working independently together in a so called data processing pipeline, that keeps up with the CT scanner's maximal frame rate of up to 8 kHz. The newly developed data processing stages are freely programmable and combinable. In order to achieve the highest processing performance all relevant data processing steps, which are required for a standard slice image reconstruction, were individually implemented in separate stages using Graphics Processing Units

*Corresponding author.
E-mail address: t.frust@hzdr.de

(GPUs) and NVIDIA's CUDA programming language. Data processing performance tests on different high-end GPUs (Tesla K20c, GeForce GTX 1080, Tesla P100) showed excellent performance.

Keywords: Computed tomography; Image reconstruction; Multithreading; Parallel algorithms; Pipeline processing; Real-time systems

---

PROGRAM SUMMARY/NEW VERSION PROGRAM SUMMARY

Program Title: GLADOS/RISA
Licensing provisions: LGPLv3
Programming language: C++/CUDA
Supplementary material: Test data set, used for the performance analysis.
Nature of problem:
Ultrafast computed tomography is performed with a scan rate of up to 8 kHz. To obtain cross-sectional images from projection data computer-based image reconstruction algorithms must be applied. The objective of the presented program is to reconstruct a data stream of around $1.3\,\mathrm{GB\,s^{-1}}$ in a minimum time period. Thus, the program allows to go into new fields of application and to use in the future even more compute-intensive algorithms, especially for data post-processing, to improve the quality of data analysis.
Solution method:
The program solves the given problem using a two-step process: first, by a generic, expandable and widely applicable template library implementing the streaming paradigm (GLADOS); second, by optimized processing stages for ultrafast computed tomography implementing the required algorithms in a performance-oriented way using CUDA (RISA). Thereby, task-parallelism between the processing stages as well as data parallelism within one processing stage are realized.

1. Motivation

The ROFEX (**RO**ssendorf ultra**F**ast **E**lectron beam **X**-ray CT) scanners, ultrafast electron beam X-ray computed tomography scanners (CT), were recently developed at Helmholtz-Zentrum Dresden-Rossendorf [1]. This technique is used for a non-intrusive investigation of rapidly moving structures in technical devices. Its most prominent application is the study of multiphase flows, which are widely found in many industrial applications, such as

in chemical reactors, heat exchangers or pipeline systems. Usually, in computed tomography a radiation source is directed to an object of investigation and the radiation is measured behind the object by a detector array. By collecting the radiation attenuation data (so-called projections) from various angular positions a data set, called sinogram, is acquired, which is subsequently used as input for a CT reconstruction algorithm. Frequently used reconstruction algorithms are the so-called filtered back-projection [2] or algebraic reconstruction techniques [3]. In this way computed tomography produces cross-sectional or volumetric images of material distributions of the scanned objects. Mostly known are CT scanners in hospitals or labs for non-destructive testing. More recently also synchrotron light sources are being used for X-ray tomography of small specimen [4, 5, 6]. Such scanners or facilities are, however, not suited for multiphase flows since neither the scanner nor the object of investigation can be rotated with the required speed. For fast dynamic processes, such as multiphase flows or biomechanical motions, blurring effects occur which deteriorate image quality significantly [7]. Additionally, using electron beam tomography there are no artificial and unwanted motions introduced, because neither the scanner nor the object of investigation need to be rotated mechanically.

In order to overcome these difficulties, the ROFEX-type scanners are operated with a focused electron beam. The beam is circularly deflected on a tungsten target and this way produces a moving X-ray source (see Figure 1). The electron beam can be deflected up to 8 kHz which leads to a corresponding frame rate. The scanners are equipped with a double layer ring detector of about 400 pixels each. The spatial resolution is about 1 mm [1]. Examples, for which this imaging technique has proven its great value are for instance the analysis of flow conditions in a gas-solid fluidized bed [9], the visualization and quantitative analysis of dispersive mixing by a helical static mixer [10] or the particle velocity measurement in spout fluidized beds [8].

Currently, the ROFEX scanners are operated in a batch mode, which is given by the current data transfer and processing architecture. A recently developed data processing toolkit that uses graphics processing units (GPUs) achieves a reconstruction frame rate of up to 140 Hz [11]. However, for the future it would be of very high value to operate the scanners in continuous mode, e.g. to have an immediate visual feedback for the operator or to use the scanner in a feed-back control loop. Especially for using the scanner in a feed-back control loop several frames are required within a single control loop to perform image analysis to come to a decision. Hence, rapid data process-

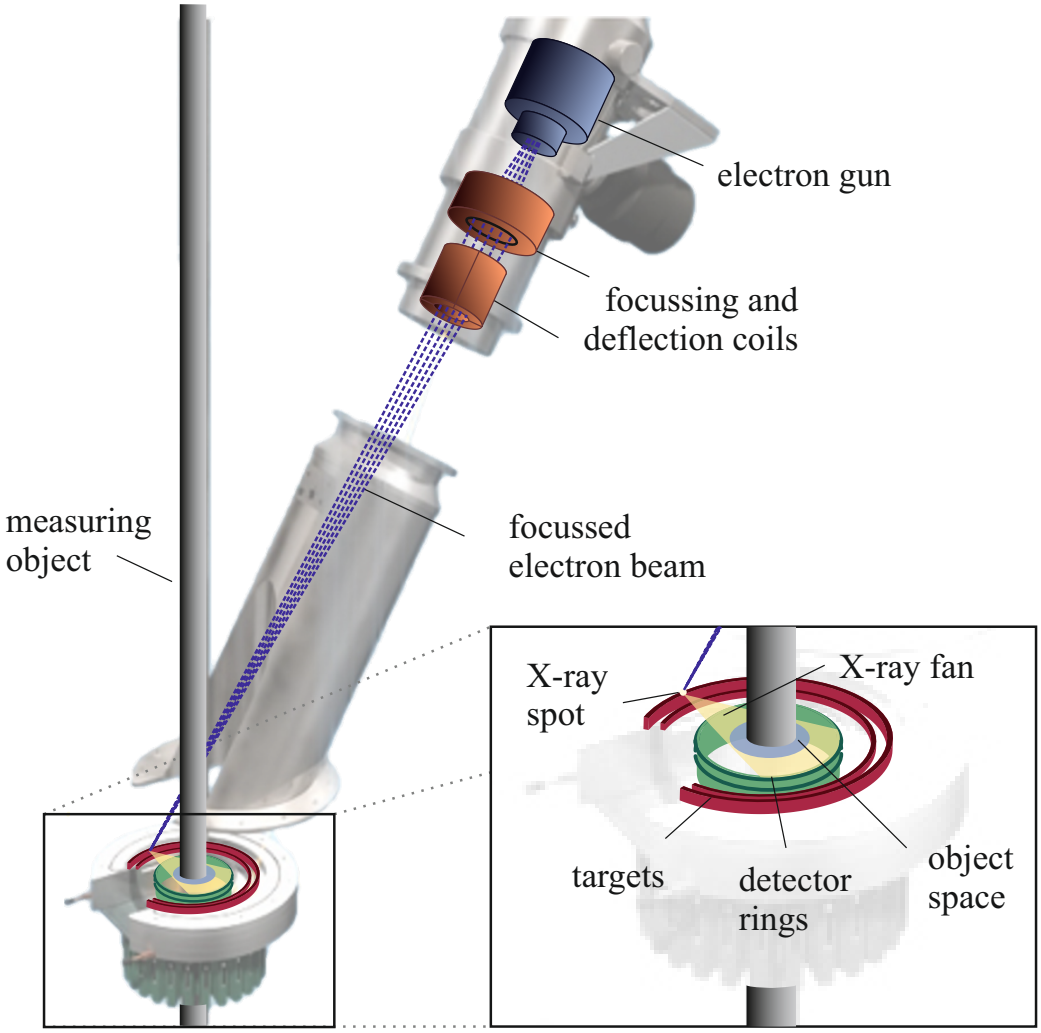

Figure 1: Sketch of the measuring principle of the ROFEX CT scanners [8].

ing has to be able to reconstruct slice images at the typical scanner operating rate between 2 kHz and 8 kHz. Finally, computationally intensive data processing algorithms, like non-linear interpolations or algebraic reconstruction algorithms, could then be implemented into the data processing pipeline and performed on powerful compute nodes to increase image reconstruction and analysis quality.

Thus, a modular and scalable data processing tool was constructed implementing the software pipelining paradigm. The implemented algorithms were accelerated using NVIDIA GPUs programmed with CUDA. In this paper related work, the architecture of the modular data processing program as well as performance results are presented.

2. Related work

Significant speedup over a parallel implementation on a single Central Processing Unit (CPU) can be achieved, when exploiting data parallelism of Graphics Processing Units (GPUs). Vázquez et al. [12, 13] presented a matrix approach to tomographic reconstruction for the filtered back projection algorithm as well as the Simultaneous Iterative Reconstruction Technique (SIRT) using sparse matrix-vector products for the back and forward projection operations. This approach shows speedup of up to factor 42 compared to the CPU implementation. Mueller et al. [14, 15] discuss the suitability of different hardware accelerators, like GPUs, Field Programmable Gate Arrays (FPGAs) or the IBM Cell B.E. architecture, for image reconstruction algorithms. They concluded that GPUs prove to be more cost-efficient than the other approaches. The filtered back projection as well as iterative reconstruction methods have been accelerated on GPUs by Diéz et al. resulting in significant speedup over their CPU implementation as well [16]. The ASTRA Toolbox [17] is a toolbox, with a MATLAB and Python interface, of high-performance GPU primitives for 2D and 3D tomography. It supports parallel and fan beam geometries with flexible source/detector geometries.

The general concept of stream processing has a long tradition as shown by the review of Stephens [18]. There exist runtime environments, as well as compiler techniques, e.g. StreamIt [19], Cg [20] or Auto-Pipe [21]. GStream [22] implements the data streaming principle for GPU clusters. A similar approach was chosen by Vogelgesang et al. [23] in the UFO framework. Data processing is modelled as the connection between processing nodes solving a specific subproblem. Furthermore, this framework implements specific al-

gorithms for image reconstruction using OpenCL, e.g. the filtered back projection [2]. Up to now, image reconstruction for the ROFEX-scanners was implemented using an in-house developed software [11]. The UFO framework served as a comparison.

None of the available frameworks implementing the streaming paradigm have become a standard in the scientific community so far. Some of them are not open source or the future support is unclear. The UFO-framework is the most related open-source library available. It is GLib-based and implemented in the C programming language. On the contrary, the approach of this publication makes use of the object-oriented programming features already being integrated in the C++11 programming standard and the C++ Standard Template Library's (STL) concurrency support instead of relying on an external library. Thus, a generic software pipeline and CUDA stages for image reconstruction were developed.

3. Analysis of the current data flow structure

The basis for successfully operating real-time scanners are data transfer and processing at scanning speed. Figure 2 shows a schematic overview of the data transfer and processing for the existing ROFEX scanners before applying the optimizations presented in this work. Up to now, the radiation detectors are sampled with a frequency of 1 MHz providing a constant data stream with a bandwidth of around $1.3\,\mathrm{GB\,s^{-1}}$ that is stored in a scanner-internal 32 GB-sized random-access memory (RAM). Thus, 25 s can be captured during one CT scan sequence. Subsequently, the detector data is downloaded from the scanner to a central data storage located in the nearby data centre before the next scan sequence can be started. Unfortunately, this data transfer process requires several minutes because of the used 1 Gbit Ethernet (GbE) network data interface. As soon as the data is available on the central data storage it is reconstructed using the reconstruction workstation which is connected to the data centre via a 10 GbE network. This is another time-consuming data transfer.

Once the entire measurement data is temporarily stored in the RAM of the workstation, the original reconstruction program processes the whole block through different processing stages. Thus, the application has a very high memory consumption on host side and furthermore, the processing latency for one sinogram is very high. The workstation is equipped with an NVIDIA Tesla K20c GPU and sufficient host memory to store the whole

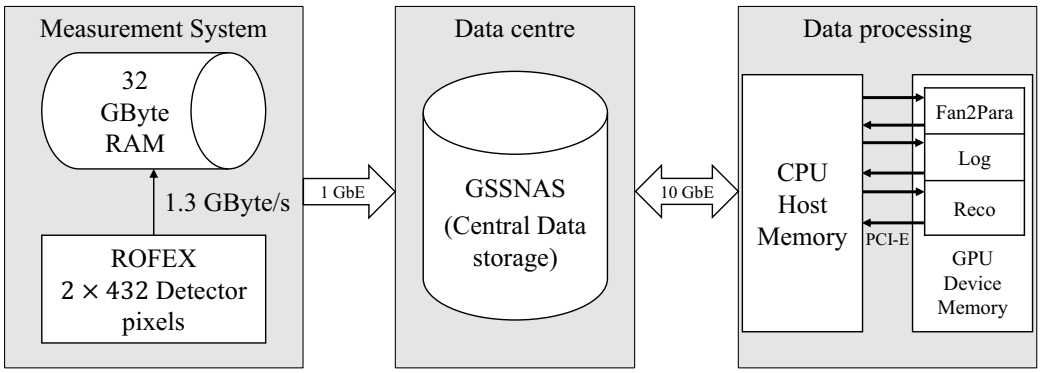

Figure 2: System overview of original data transfer and processing consisting of the measurement system, the data centre and the reconstruction workstation.

measurement in the RAM of the workstation. Unfortunately, not all data processing algorithms are implemented for GPU usage and thus, multiple data transfers from host to device and vice-versa are required increasing data processing time again.

Altogether, the original data transfer and processing are not suited for online or efficient application on GPU compute nodes due to the architecture of the interconnection network and the design of the reconstruction program itself. Therefore, the introduction of a new interconnection structure and the implementation of a data processing tool being able to process a data stream in real-time is mandatory.

4. Implementation of a real-time data processing software

4.1. GLADOS - Generic Library for Asynchronous Data Operations and Streaming

Image processing (or rather stream processing in general) can be perceived as a sequence of basic operations. For example, a simple image filtering application can be divided into three tasks: loading data, filtering the image and saving the result. By applying the so-called pipeline pattern [24] each subproblem can be solved independently; these subproblems are called stages. This program structure is applicable to a variety of problems requiring stream processing. In order to solve these problems in a straightforward way, independently from the required algorithms, the Generic Library for

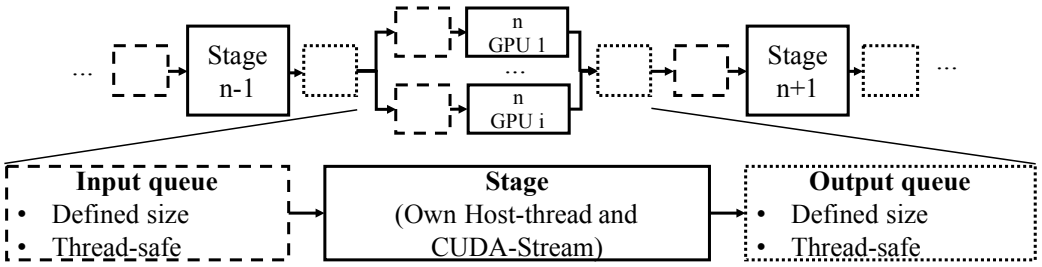

Figure 3: Schematic diagram of the data processing pipelines' structure. Stages communicate through thread-safe input and output queues.

Asynchronous Data Operations and Streaming (GLADOS) was developed. It implements the pipeline pattern in a generic and portable way.

When working with heterogeneous systems, parallelism can be exploited on multiple levels: data and task parallelism. Especially, when the problem size is comparatively small, a single function is not able to fully utilize the whole device. Thus, creating a software pipeline offers the potential to increase data throughput significantly. There are two options: task parallelism between a) different input data or b) various subproblems. In the first case, device functions can be called without any synchronization on host side minimising the overhead. By using dynamic parallelism, a feature being available since OpenCL 2.0 and CUDA 5.0, device functions can be launched from threads running on the device. By using this function the overhead from calling device functions on the host side can be decreased even further. The drawback of that approach is a limited handling in terms of expandability and usability. In the second case the processing chain is clearly divided into several subproblems that can easily be connected with each other. Thus, approach b) promises a superior expandability and usability and was, therefore, chosen to be realized.

GLADOS is implemented as a header-only library. Internally, it makes use of C++11 and the concurrency support of the C++ STL. Figure 3 depicts a schematic diagram of the pipeline's structure as provided by GLADOS. The core element is the stage. A stage is executed asynchronously with regard to the main thread and the other stages and solves a specific subproblem (e.g. image filtering). Each stage is composed of three parts: an input side, an output side and an implementation.

The input and output side are responsible for inter-stage data exchange only. They are connected with their respective counterparts of other stages. For example, the output side of a stage which loads image data is connected with the input side of the next stage which processes data. Data is transferred from one stage to another using C++11 move semantics to prevent data copies. An input side additionally stores the incoming data in a thread-safe FIFO kind of way and makes it available to the underlying stage. The user needs to declare the incoming and outgoing data types, routines for handling the input and output callbacks and a run method which is the stage's entry point in the implementation of a stage. Other than that, the implementation can be designed on the user's choice.

During data processing the types of data may change, e.g. from single to double precision. This usually requires memory allocation and has to be prevented, especially in CUDA, since memory allocations require a synchronization of all used threads. In this case, memory transfer operations and kernel executions would not overlap and data processing would be interrupted. Thus, a memory pool is introduced to manage the allocation of memory in the constructor of all used device data structures. In this way, maximum device task concurrency is ensured and data processing throughput increases.

Last but not least, multiple compute devices can be used by duplicating the stages for each available device. In this case, one stage schedules input data statically or dynamically between the accelerators. This way, the application can be used on hardware with multiple heterogeneous compute devices.

4.2. RISA - ROFEX in-situ analysis

The ROFEX scanners have a special geometry. Thus, data preprocessing is required to transform detector data into a sinogram in order to be able to apply standard reconstruction algorithms, such as the filtered back projection [2]. Figure 4 shows the standard image reconstruction pipeline for ROFEX data. First, there is an input stage, which loads the sinograms from hard disk storage or receives them from an interconnection network. As soon as the copy stage schedules and transfers data from host to device, all processing steps are implemented on GPUs using CUDA.

The raw data stream is not ordered as required by the successive stages. Thus, data is restructured into sinograms ordered by detector pixels and projections (Figure 4 a) to simplify the subsequent operations and to make

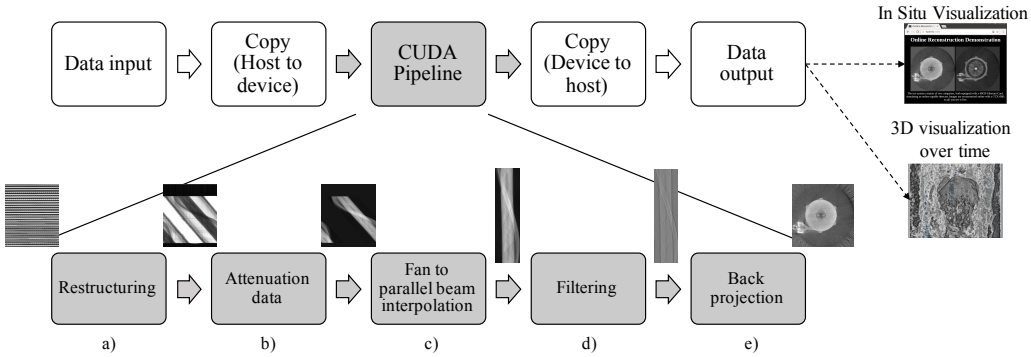

Figure 4: The implemented ROFEX image processing pipeline with data preprocessing and reconstruction from parallel beam sinogams (grey represents stages accelerated using CUDA).

their implementation more straightforward. Therefore, this stage computes a lookup table at program initialization storing the relation between unstructured and restructured data. The CUDA kernel spans $N_{\text{fanDet}} \times N_{\text{fanProj}}$ CUDA threads (N_{fanDet} being the number of detectors and N_{fanProj} being the number of projections in the fan beam sinogram). Each CUDA thread copies one element from the unordered to the ordered sinogram using the precomputed lookup table. It would have been possible to integrate the operation into the existing fan to parallel beam rebinning stage after computing the attenuation data, because both use a lookup table. Nevertheless, it is to be expected that this operation is relatively cheap compared to the other stages and thus, does not have a negative impact on the overall performance.

To compute the attenuation data (Figure 4 b) a dark measurement data set with values I_{dark} (electron beam is off, only the background noise of the detectors is measured) and a reference measurement data set with values I_{ref} (measurement of the base substance, e.g. fluid, without the test subject) need to be captured beside the data set I_{data} for the real measurement. The attenuation value E for the pixel with the coordinates i, j is given in equation 1. It is mapped into CUDA by creating one CUDA thread for each pixel computing equation 1 independently.

$$E_{\text{ij}} = \log\left(\frac{I_{\text{ref,ij}} - I_{\text{dark,ij}}}{I_{\text{data,ij}} - I_{\text{dark,ij}}}\right) \tag{1}$$

The fan beam to parallel beam interpolation stage (Figure 4 c) creates a

parallel beam sinogram, which also gives a more homogeneous noise distribution in the reconstructed image compared to the direct reconstruction from fan beam geometry [25]. It is possible to solve the image reconstruction problem, without using this stage by using an algebraic reconstruction technique with a direct application of the geometry. Nevertheless, this approach is not promising when regarding performance because the analysis will show, that in fact the back projection implementation is the bottleneck of the presented application. This fan to parallel beam interpolation stage is implemented in CUDA using another lookup table created at program initialization. It stores the relations between the fan and user-definable virtual parallel beam geometry. The CUDA kernel spans $N_{\text{parDet}} \times N_{\text{parProj}}$ CUDA threads, with N_{parDet} being the number of detectors and N_{parProj} being the number of projections in the parallel ray sinogram over 180 degrees. Each CUDA thread computes one pixel in the parallel beam sinogram independently. Thus, N_{parDet} and N_{parProj} can be used to balance between image quality and the required reconstruction rate. A symmetry property is valid, given in equation 2, for parallel projections $p(t, \phi)$, where t represents the orthogonal distance of the ray to the iso-center and ϕ the angle between the ray and the abscissa.

$$p(t, \phi + \pi) = p(-t, \phi) \tag{2}$$

By averaging rays measured twice at the opposite of the object, the CUDA kernel transforms the projections distributed over 360° into a parallel beam sinogram from 0° to 180° without loss of information.

The reconstruction procedure using the filtered back projection is split into two stages: filtering (Figure 4 d) and back projection (Figure 4 e). Each projection is transformed into the Fourier space via the Fast Fourier Transform (FFT), weighted with a filter function and inversely transformed via the inverse FFT. The FFTs are implemented using the cuFFT-library [26]. When implementing the back projection operation, there generally exist two approaches: a ray driven and a pixel driven approach. The first one follows the ray from a discrete projection and samples it into equally spaced parts. At each sample it distributes the intensity of the projection over the neighbouring pixels using a 2D-interpolation. The second approach starts at the centre of a pixel in the reconstruction grid. From this point the intersection with the discrete projections is determined by following the ray path. The intersection does not necessarily line up with the discrete projections. Thus, some kind of 1D-interpolation needs to be performed. In this case, the pixel driven back projection method was chosen. It is much more suitable for

data-parallel execution compared to the ray driven approach. Furthermore, it is computationally less expensive (1D- vs. 2D-interpolation) [25]. The implemented CUDA-kernel spans $N_{\text{pixel}} \times N_{\text{pixel}}$ CUDA threads. Each CUDA thread computes the attenuation coefficient for one pixel in the reconstruction grid. There are two different CUDA kernels implemented computing the back projection algorithm: one applies a nearest-neighbour interpolation using texture fetches and the second a linear interpolation.

```
1  auto pipeline = glados::pipeline::Pipeline{};
2  // create stages
3  auto filter = pipeline.create<filter_stage>(configFile);
4  //...
5  //connect stages
6  pipeline.connect(source, h2d);
7  pipeline.connect(h2d, filter);
8  pipeline.connect(filter, backproject);
9  pipeline.connect(backproject, d2h);
10 pipeline.connect(d2h, sink);
11 //run the stages
12 pipeline.run(source, h2d, filter, backproject, d2h, sink);
13 //wait for termination
14 pipeline.wait();
```

Listing 1: The implementation of the filtered back projection algorithm using GLADOS and RISA in C++.

The implemented stages can then be connected to a processing pipeline using GLADOS. Listing 1 shows the implementation of the filtered back projection algorithm using GLADOS and RISA in C++ as an example. This application can now easily be extended or adapted by creating (lines 3-5), connecting (lines 6-10), running (line 12) and waiting (line 14) for additional or different stages. The termination of the program is realized using a sentinel image sent out by the source stage. The sentinel is propagated from one stage to another signalling the termination automatically. As soon as all stages are terminated the application advances after line 14 in the given example. Hence, the user does not necessarily require an understanding of

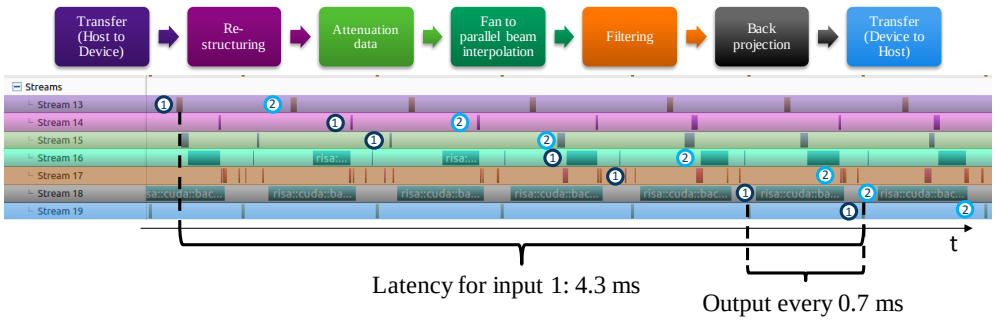

Figure 5: The time line of device functions captured with the NVIDIA Visual Profiler. Memory transfer operations and CUDA kernels overlap. The markers 1 and 2 identify two subsequent sinograms passed from one stage to another ($N_{\mathrm{parDet}} = 256, N_{\mathrm{parProj}} = 1024, N_{\mathrm{pixel}} = 256$).

the concrete implementation of the algorithms to configure a custom data processing pipeline.

5. Performance analysis

The performance of the implementation is evaluated on multiple systems and GPUs to quantify the scalability between different as well as multiple GPUs. Measurements are taken on an Ubuntu 16.04 workstation equipped with two Intel Xeon E5-2637 v3 CPUs (each with four physical cores at 3.5 GHz and active multithreading), 16 GiB RAM, the founders edition of an NVIDIA GeForce GTX 1080 and an NVIDIA Tesla K20c. The code is compiled with GCC 5.4 and CUDA 8. The analysis is performed with a test data set consisting of 432 detector pixels and 500 projections per fan beam sinogram captured at 2 kHz electron beam deflection frequency. The input data type is 16 bit unsigned integer which is converted to single precision floating point numbers during the computation of attenuation data. The back projection kernel with linear interpolation was applied for all measurements.

In order to meet and exceed the required frame rates device utilization was maximised using data and task parallelism. Figure 5 presents the time line of the presented application in steady state for the configuration $N_{\mathrm{parDet}} = 256, N_{\mathrm{parProj}} = 1024$ and $N_{\mathrm{pixel}} = 256$ performed on a

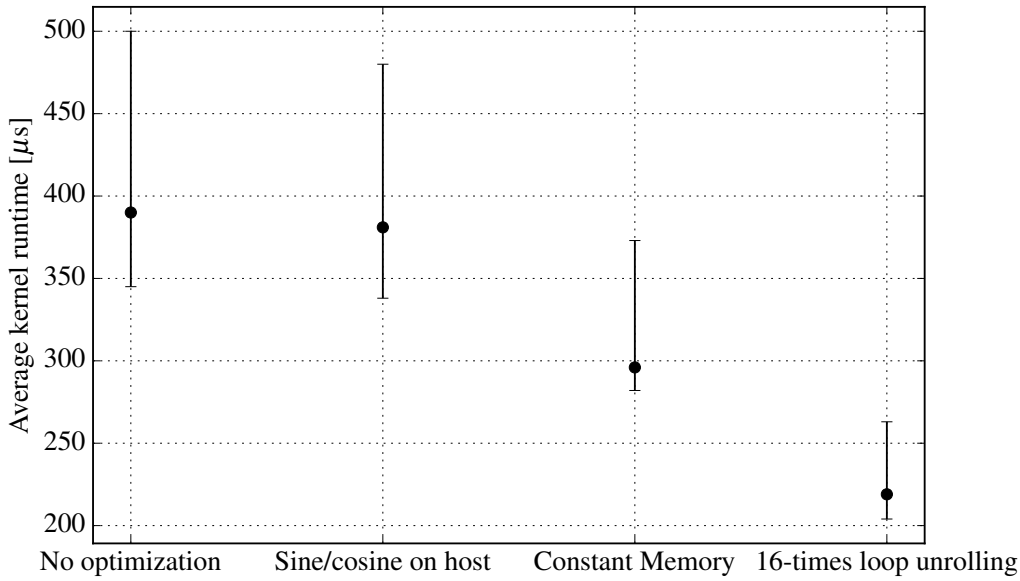

Figure 6: Average kernel runtime of the back projection operation for different optimization strategies ($N_{\mathrm{parDet}} = 256, N_{\mathrm{parProj}} = 512, N_{\mathrm{pixel}} = 256$, GeForce GTX 1080).

single GeForce GTX 1080 captured with the NVIDIA Visual Profiler. Each coloured line corresponds to one processing stage. The dark blocks identify kernel executions respectively data transfer. It can be recognized that memory transfer operations as well as CUDA kernels overlap. The pauses between two consecutive executions of the back projection kernel appear to origin from an overhead on host side. Furthermore, the software pipeline's structure can be clearly identified, following the pass of CT data sets from one stage to another. Therefore, two markers 1 and 2 are added in the profiler output identifying two consecutive sinogram data sets. The corresponding data transfers and kernel executions were identified by having a look at the kernel invocation time line on host side, which allows to link an image to its associated kernel execution on the GPU device. It shows again, that multiple images are processed simultaneously. The latency for sinogram 1 is 4.3 ms, whereas all 0.7 ms one element is transferred from GPU to CPU memory.

The identification and removal of bottlenecks is mandatory when applications require the highest performance. In this application the back projection operation was identified as the most time-consuming part (see Figure 5).

Thus, an additional focus was directed to its optimization. Three strategies have been analysed: a) the computation of the sine and cosine values on the host side, b) the use of the architecture-specific constant memory for these values and c) the unrolling of the loop over the projections. The strategy of pre-computation was promising because the geometry and configuration of the system is constant for a whole measurement. Thus, the computation on host side and the required data transfer only needs to be executed at program initialization. The computation of sine and cosine values on host in combination with the use of constant memory improves the runtime by a factor of 1.3 (see Figure 6). Thereof, 16-times loop unrolling enhances the runtime by the factor of 1.3 again and, finally, the overall improvement factor is approximately 1.8. A commonly used metric for comparing the back projection performance is given by the so-called Giga-Updates-Per-Second (GUPS) metric, first introduced by Goddard et al. [27] (see equation 3).

$$GUPS = \frac{N_{pixel}^2 \cdot N_{proj}}{1024^3 \cdot \bar{t}_{bp,kernel}} \tag{3}$$

The measured GUPS values for the given back projection implementation on different hardware configurations are given in Figure 7. For a small pixel-grid size ($N_{\text{pixel}} < 160$) the GUPS metric increases linearly, the kernel is memory-bound. For a higher resolution grid, the kernel is compute-bound as shown by the GUPS metric approximately being constant.

The maximum achievable reconstruction rate is depicted in Figure 8 for a constant size of the parallel ray sinogram ($N_{\text{parDet}} = 256, N_{\text{parProj}} = 512$) as a function of the reconstruction grid size. Regarding image quality there is no additional benefit of increasing this pixel grid size above $N_{\text{pixel}} = 256$. The scanners have a maximum spatial resolution of 1 mm at an image size of about 200 mm. Choosing a convenient value for N_{pixel} is mandatory for gaining the best performance. The measurement was taken on a single GeForce GTX 1080 as well as a single Tesla K20c. Furthermore, reconstruction rates were measured using both cards simultaneously with static scheduling. For all tested configuration options the GeForce GTX 1080 is able to meet and exceed the typical scan rate of 2 kHz (see highlighted dashed line). Tesla K20c's rates are lower by a factor of approximately 3 to 4. In this case, the 2 kHz could not be reached. The Tesla K20c has a 2.5 times lower theoretical peak performance in single precision compared to the GeForce GTX 1080. Furthermore, the GPU device memory bandwidth of the GTX 1080 is higher

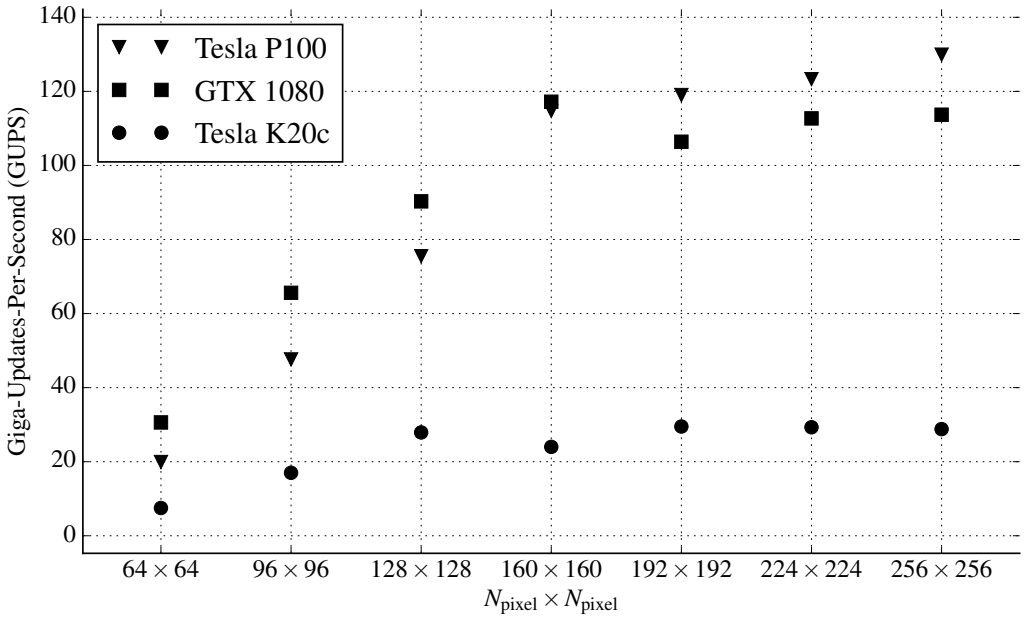

Figure 7: The measured Giga-Updates-Per-Second metric for the back projection kernel with linear interpolation on different GPU hardware, with $N_{\text{parDet}} = 256 \times N_{\text{parProj}} = 512$ and variable pixel grid N_{pixel}.

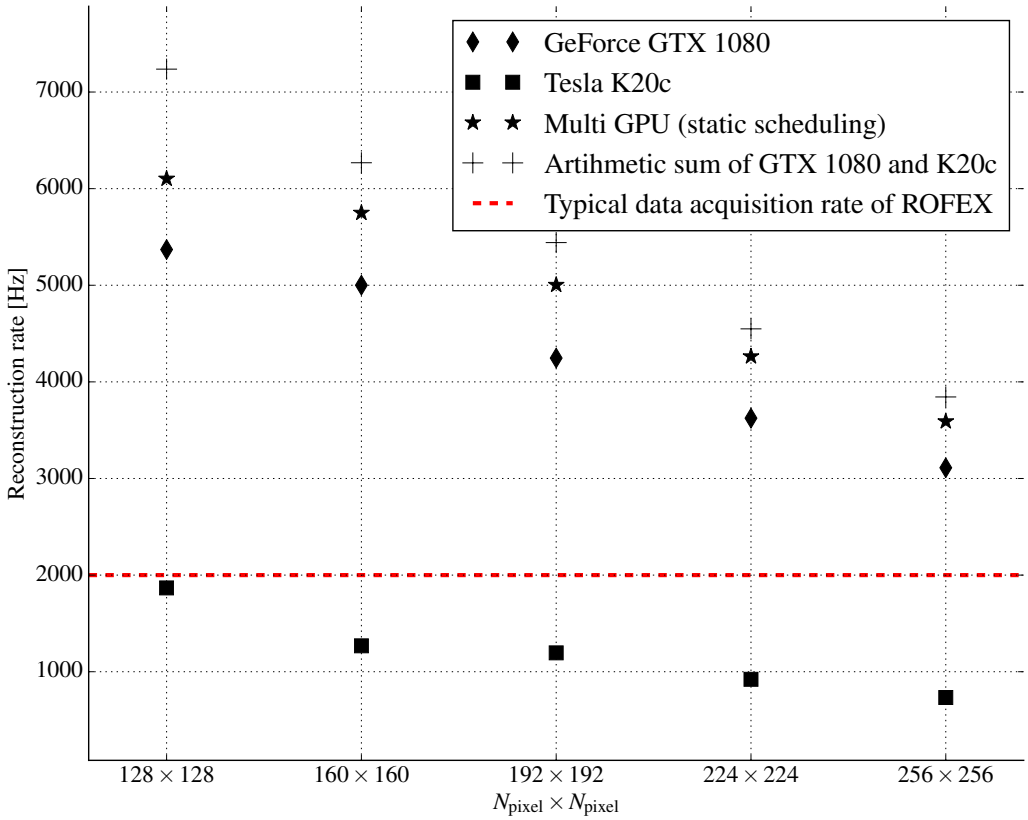

Figure 8: Reconstruction rates for one GeForce GTX 1080 and one Tesla K20c, as well as for both cards simultaneously with static scheduling for different resolutions in the reconstructed image ($N_{\text{parDet}} = 256, N_{\text{parProj}} = 512$).

by a factor of 1.5 compared to the Tesla K20c. If the measured metrics are compared with these theoretical quantities, the performance difference of factor 3 to 4 is reasonable. The application is also able to use multiple heterogeneous accelerators simultaneously as shown by the reconstruction rates in Figure 8. By applying a static scheduling using a round robin strategy in the host to device copy stage, the obtained reconstruction rates increase. The scheduling method uses an empirically determined weighting factor basing on the performance capability of the available GPUs in the system to distribute input data.

Furthermore, the application was also tested on JURON, one of two pilot systems developed by IBM and NVIDIA in the Pre-Commercial Procurement during the Human Brain Project (HBP) Ramp-up phase. It is located at Juelich Supercomputing Centre (JSC). It uses IBM Power8' processors and four NVIDIA Tesla P100 accelerators interconnected via NVLink on one IBM Power S822LC for HPC (also known as IBM Minsky) compute node. Figure 9 shows the measured reconstruction rates on a single node of this system. First, it demonstrates that the maximum rate of 8 kHz can be reached or even exceeded when using multiple GPUs. Second, the application does not scale linearly. For each additional GPU all stages are duplicated and thus, the number of host threads increases. At the same time, the number of available physical cores in the system stays constant. Hence, it is more difficult for the scheduler of the operating system to distribute the host threads over the available resources. The overhead to transfer elements between stages increases as well. Third, the performance difference between a single Tesla P100 and a single GeForce GTX 1080 is comparably marginal. This is constituted in the implemented stages performing floating point operations in single precision. In single precision the theoretical floating point performance of a single Tesla P100 is 10.6 TFLOPS/s compared to 8.9 TFLOPS/s of a single GeForce GTX 1080. This difference is negligible explaining the marginal performance difference for our specific application. Nevertheless, a careful investigation is required if the additional features of the Tesla series can be neglected in a specific use case. Furthermore, this fact cannot easily be transferred to a system with multiple GPUs without regarding the capabilities of the overall system.

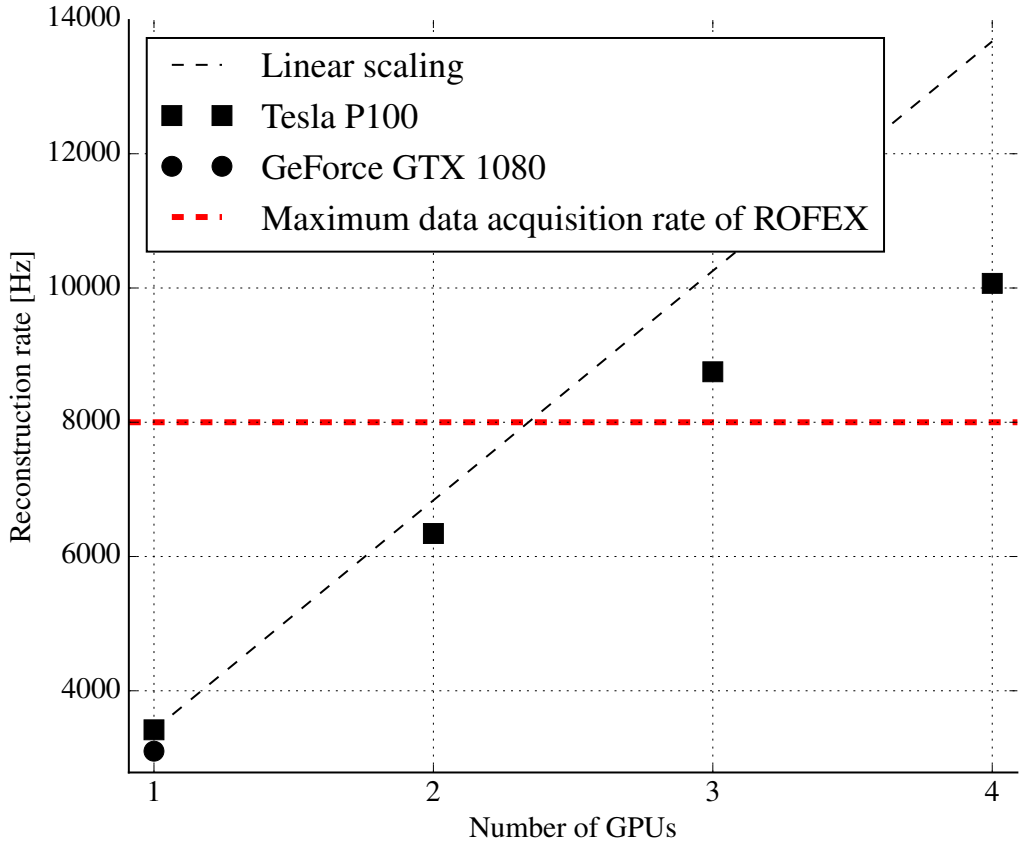

Figure 9: Measured reconstruction rates as a function of the number of used GPUs on one IBM Power S822LC for HPC node at Juelich Super Computing Center. ($N_{\mathrm{parDet}} = 256, N_{\mathrm{parProj}} = 512, N_{\mathrm{pixel}} = 256$)

6. Summary and outlook

In this paper, a modular and scalable data processing pipeline program for image reconstruction for ultrafast electron X-ray CT was presented and analysed for its scalability between different as well as multiple GPUs. Therefore, a generic and universally applicable template library was used implementing the pipeline pattern. Another application-specific library, called RISA, was presented implementing the required processing stages for image reconstruction of ROFEX in CUDA. Finally, an application program was compiled connecting the implemented processing stages to a pipeline using both libraries.

The performance analysis showed reconstruction rates of more than 2 kHz for single GPU usage. By using multiple GPUs the maximum frame rate of ROFEX at 8 kHz could even be exceeded. Thereby, the scalability of the application between different as well as multiple GPUs could be proven. Hence, visual inspection and active process feedback control are now attainable. Furthermore, the reduced time-to-solution allows the utilization of more complex algorithms for post-processing on GPU clusters, e.g. iterative reconstruction algorithms or more compute-intensive interpolation methods that will improve the quality of the slice images and thus, the content of scientific results.

Both the most up to date versions of GLADOS and RISA source code are available at `https://github.com/HZDR-FWDF/`[28].

Acknowledgement

The authors thank the Dresden GPU Center of Excellence (`http://gcoe-dresden.de`) for the access to an NVIDIA Tesla K20c as well as the Human Brain Project (`https://www.humanbrainproject.eu/`) for access to an IBM Minsky Node.

The authors acknowledge the Helmholtz Association for support of the research within the frame of the Helmholtz Energy Alliance 'Energy Efficient Chemical Multiphase Processes'.

7. References

[1] F. Fischer, U. Hampel, Nuclear Engineering and Design 240 (2010) 2254 – 2259. doi:`10.1016/j.nucengdes.2009.11.016`.

[2] A. C. Kak, M. Slaney, Principles of Computerized Tomographic Imaging, IEEE Press, 1988.

[3] R. Gordon, R. Bender, G. T. Herman, Journal of Theoretical Biology 29 (1970) 471 – 481. doi:10.1016/0022-5193(70)90109-8.

[4] A. Rack, S. Zabler, B. Müller, H. Riesemeier, G. Weidemann, A. Lange, J. Goebbels, M. Hentschel, W. Görner, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 586 (2008) 327 – 344. doi:10.1016/j.nima.2007.11.020.

[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. Danilewsky, T. Waterstradt, W. Diete, H. Riesemeier, B. Müller, T. Baumbach, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 267 (2009) 1978 – 1988. doi:10.1016/j.nimb.2009.04.002.

[6] O. H. Seeck, C. Deiter, K. Pflaum, F. Bertam, A. Beerlink, H. Franz, J. Horbach, H. Schulte-Schrepping, B. M. Murphy, M. Greve, O. Magnussen, Journal of Synchrotron Radiation 19 (2012) 30–38. doi:10.1107/S0909049511047236.

[7] G. A. Johansen, U. Hampel, B. T. Hjertaker, Applied Radiation and Isotopes 68 (2010) 518 – 524. doi:10.1016/j.apradiso.2009.09.004.

[8] M. Bieberle, F. Barthel, Chemical Engineering Journal 285 (2016) 218 – 227. doi:10.1016/j.cej.2015.10.003.

[9] M. Bieberle, F. Barthel, U. Hampel, Chemical Engineering Journal 189–190 (2012) 356 – 363. doi:10.1016/j.cej.2012.02.028.

[10] S. Rabha, M. Schubert, F. Grugel, M. Banowski, U. Hampel, Chemical Engineering Journal 262 (2015) 527 – 540. doi:10.1016/j.cej.2014.09.019.

[11] A. Bieberle, T. Frust, M. Wagner, M. Bieberle, U. Hampel, Flow Measurement and Instrumentation 53, Part A (2017) 180–188. doi:10.1016/j.flowmeasinst.2016.04.004.

[12] F. Vázquez, E. Garzón, J. Fernández, Journal of Structural Biology 170 (2010) 146 – 151. doi:10.1016/j.jsb.2010.01.021.

[13] F. Vázquez, E. M. Garzón, J. J. Fernández, The Computer Journal 54 (2011) 1861–1868. doi:10.1093/comjnl/bxr033.

[14] K. Mueller, F. Xu, N. Neophytou, Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?, 2007. doi:10.1117/12.716797.

[15] F. Xu, K. Mueller, Physics in Medicine and Biology 52 (2007) 3405.

[16] D. C. Díez, H. Mueller, A. S. Frangakis, Journal of Structural Biology 157 (2007) 288 – 295. doi:10.1016/j.jsb.2006.08.010.

[17] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. D. Beenhouwer, K. J. Batenburg, J. Sijbers, Opt. Express 24 (2016) 25129–25147. doi:10.1364/OE.24.025129.

[18] R. Stephens, Acta Informatica 34 (1997) 491–541. doi:10.1007/s002360050095.

[19] W. Thies, M. Karczmarek, S. Amarasinghe, StreamIt: A Language for Streaming Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 179–196. doi:10.1007/3-540-45937-5_14.

[20] W. R. Mark, R. S. Glanville, K. Akeley, M. J. Kilgard, ACM Trans. Graph. 22 (2003) 896–907. doi:10.1145/882262.882362.

[21] R. D. Chamberlain, M. A. Franklin, E. J. Tyson, J. H. Buckley, J. Buhler, G. Galloway, S. Gayen, M. Hall, E. F. B. Shands, N. Singla, Computer 43 (2010) 42–49. doi:10.1109/MC.2010.62.

[22] Y. Zhang, F. Mueller, in: 2011 International Conference on Parallel Processing, pp. 245–254. doi:10.1109/ICPP.2011.22.

[23] M. Vogelgesang, S. Chilingaryan, T. d. Santos, A. Kopmann, in: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, pp. 824–829. doi:10.1109/HPCC.2012.116.

[24] T. Mattson, B. Sanders, B. Massingill, Patterns for Parallel Programming, first ed., Addison-Wesley Professional, 2004.

[25] J. Hsieh, Computed tomography : principles, design, artifacts and recent advantages, SPIE Press, 2015. doi:10.1117/3.2197756.

[26] NVIDIA, CuFFT Library User's Guide, 2016. URL: http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf.

[27] I. Goddard, A. Berman, O. Bockenbach, F. Lauginiger, S. Schuberth, S. Thieret, in: Proc. SPIE 6498, Computational Imaging V, volume 6498, pp. 64980R–64980R–8. doi:10.1117/12.722160.

[28] Github repository for the presented libraries., 2017. URL: https://github.com/HZDR-FWDF.