Modeling and Optimizing Data Transfer in GPU-Accelerated Optical Coherence Tomography

Tobias Schrödter[1,2(✉)], David Pallasch[2], Sandra Wienke[3], Robert Schmitt[4], and Matthias S. Müller[3]

[1] RWTH Aachen University, Aachen, Germany
tobias.schroedter@rwth-aachen.de
[2] Fraunhofer-Institute for Production Technology IPT, Aachen, Germany
[3] IT Center, RWTH Aachen University, Aachen, Germany
{wienke,mueller}@itc.rwth-aachen.de
[4] Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University, Aachen, Germany

Abstract. Signal processing of optical coherence tomography (OCT) has become a bottleneck for using OCT in medical and industrial applications. Recently, GPUs gained more importance as compute device to achieve video frame rate of 25 frames/s. Therefore, we develop a CUDA implementation of an OCT signal processing chain: We focus on reformulating the signal processing algorithms in terms of high-performance libraries like CUBLAS and CUFFT. Additionally, we use NVIDIA's stream concept to overlap computations and data transfers. Performance results are presented for two Pascal GPUs and validated with a derived performance model. The model gives an estimate for the overall execution time for the OCT signal processing chain, including compute and transfer times.

Keywords: GPU · OCT · Performance model · CUDA

1 Introduction

Tomographic imaging methods are of great importance in medical and industrial contexts. In medicine, one focus lies on imaging quality and processing speed, whereas in industry the possibilities for automation and cost efficiency are crucial. These various requirements have led to the development of a variety of different tomographic imaging techniques. Originating from ophthalmology, one of the techniques which gained importance in the last 25 years, is optical coherence tomography (OCT). Due to its resolution in the lower micrometer range, it is used in production metrology, i.e., for measuring coating thickness in terms of quality assurance. At the Fraunhofer IPT OCT systems for medical and industrial applications are developed, including the OCT itself as well as the corresponding signal processing.

G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 421–433, 2019.
https://doi.org/10.1007/978-3-030-10549-5_33

For monitoring processes in a production or biomedical environment, not only a high spatial resolution but also a high temporal resolution is needed. Currently, this is limited by the signal processing time for OCT systems. The goal is to achieve a frame rate of 25 frames/s, which corresponds to 40 ms of processing time. Increasing the frame rate beyond this value may not benefit the human eye while observing the process, but creates headroom for further image processing and evaluation, as well as accelerating processing volumetric data. GPUs with the possibility of executing massively parallel computations promise processing of high-resolution OCT images with video frame rate. Hence, we developed a GPU-parallel CUDA version of the signal processing based on an existing CPU implementation. Since the investigated OCT system has been designed to be cost-effective, we focused solely on consumer GPUs with Pascal architecture. For testing and validating the GPU implementation a middle class and a high-end GPU have been used. To validate the performance of our implementation, we derive a performance model for the OCT signal processing chain that covers runtime prediction of kernels and data transfers. This model can also be used to get an estimate which resolutions could be achieved with a given GPU.

Thus, our main contributions are:

- A CUDA-based GPU implementation of the OCT signal processing chain with focus on leveraging highly-optimized (BLAS) libraries
- A performance model which describes the computation and the data transfer of the GPU signal processing
- Investigation of two different NVIDIA GPUs in comparison to two CPU-parallel versions

The paper is structured as follows: Sect. 2 presents related work. The basics of OCT and the used signal processing functions are described in Sect. 3. The parallelization concepts using CUDA follow in Sect. 4. In Sect. 5, we derive a performance model for the given OCT signal processing chain. The performance results of our parallelization is presented in Sect. 6. Finally, we conclude and give a short outlook in Sect. 7.

2 Related Work

Due to the needed processing time, using GPUs has become an important factor during the development of OCT systems. So far, the main focus has lied on resampling the data [11,18], or using multiple GPUs one for computation and one for visualization [15,17]. These works do not elaborate on their strategies for implementing a GPU version of the OCT signal processing chain, especially leveraging BLAS libraries have not been reported yet. In addition, different libraries are available which are designed for tomographic signal processing such as the ASTRA Toolbox [1]. However, they do not include algorithms that are specifically needed for the signal processing of the Fraunhofer IPT OCT system.

While different techniques exist for performance models for GPUs [12], we focus on analytical-based models that illustrate a comprehensive approach. That

means the model must cover the computational part, as well as the GPU-CPU data transfers, and should not be tied to a specific application. Various models have been established to describe the transfer times on distributed-memory systems [2–4]. The model of Boyer et al. [3] gives an estimate for the transfer time based on the GPU's bandwidth. A theoretical performance model, based on the Roofline Model [16], for GPU applications has been introduced as the so-called Boat Hull Model [13]. It gives an estimate of the floor of the attainable runtime for the computational part depending on the device specifications and some algorithm specific characteristics. To model concurrent copy and compute operations, Gómez-Luna et al. [8] have focused on (today) older NVIDIA architectures, where Werkhoven et al. [14] update this approach to (more) recent NVIDIA architectures with multiple copy engines. In this work, we combine the approaches of [3,13], and [14] to derive a complete model for the OCT signal processing chain as real-world application.

3 Optical Coherence Tomography

Optical coherence tomography is a cross-sectional tomographic imaging method and is mostly applied in ophthalmology, due to the possibility of creating contact free cross-sectional images of the eye [9]. In contrast to traditional imaging techniques, OCT offers a higher penetration depth than confocal microscopy and a better resolution than ultrasound imaging. Due to its destruction-free nature and the possibility of a complete fiber-optic setup, OCT has recently been introduced into new fields in biomedical imaging like cancer detection and tissue engineering, as well as in production technology for quality assurance.

OCT is an interferometric measurement technique, in which low-coherent light excites the material under investigation. Light, which is backscattered at different depths inside the sample, is overlapped with light from a reference path and interferes at the detector. The interference creates a modulated signal whose frequency depends on the depth of the reflection. In a Frequency-Domain-OCT (FD-OCT), a spectrometer is used to measure the interference and therefore obtain spectral resolved modulated data. The depths of the reflection can be computed by using a Fourier transformation on the modulated data, resulting in the depth profile of the sample. As this is only a brief overview of OCT we recommend [7] for further reading.

The OCT signal processing chain, as implemented in the OCT software at Fraunhofer IPT, is shown in Fig. 1. For controlling the line scan camera and the data acquisition boards only C++ interfaces are available, hence the driver is implemented in C++. The obtained signal is stored in vectors as `unsigned short` values and is processed serially on the CPU. From the line scan camera of the spectrometer the recorded spectrum of a 2D-scan (B-scan) is continuously written into the acquisition buffer. In the next step the data is processed and the results are written to the display buffer.

Due to physical effects, the modulated data recorded by the line scan camera is not equidistantly spaced [10]. Since Fourier transformations can only be

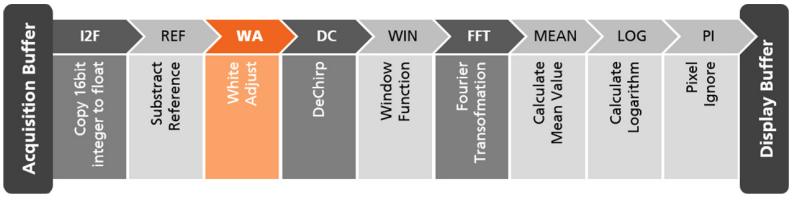

Fig. 1. Signal processing chain. Dark gray items display necessary operations of OCT signal processing, orange the bottleneck of the CPU implementation (`WA`).

applied if the data is sampled in an equidistant manner, the data must be rescaled (`DC`). To obtain the depth profile of the sample, the Fourier transformation (`FFT`) is applied. For this, we use the FFTW algorithm [6], in particular the real to complex transformation, where only the magnitude of the result is of interest. To reduce noise in the detected spectrum, the average of 1D-scans (A-scan) contained in the B-scan is computed and subtracted from the complete scan, here referred by white adjust (`WA`). Its implementation will be explained in Sect. 4.

As test data sets, we use different real-life OCT images that contain B-scans of a pill (1120×256 px, 1120×500 px) and cancerous tissue (2048×512 px). Since we are also interested in the performance for large data sets, one data set was artificially enlarged, by doubling the input data (up to 2048×8192 px). With these data sets also the correctness of the GPU implementation is assured.

In Fig. 2 the absolute runtimes of the serial reference OCT implementation are shown. The white adjust needs up to 40% of the execution time. `FFT`, `DC` and `MEAN` are further bottlenecks in the application. The red-dotted line indicates the time limit for processing data with a video frame rate. A B-scan of size 2018×1536 px is already too large to be processed in less than 40 ms. As the Fraunhofer IPT targets at larger data sets, a faster processing is needed.

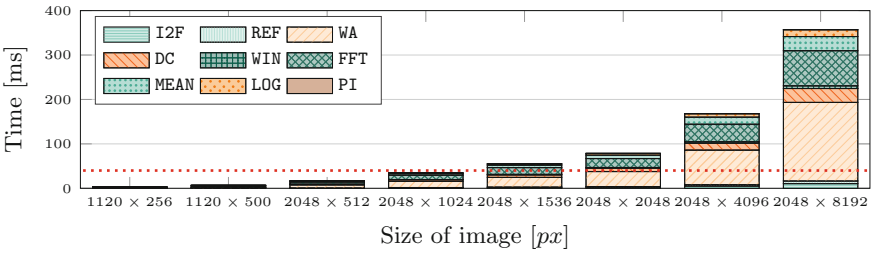

Fig. 2. Runtime of the reference implementation, split into the different processing steps. The red line indicates the target frame rate of 25 frames per second.

4 Parallelization with CUDA

Based on the existing serial CPU signal processing chain, we developed a massively-parallel GPU implementation with CUDA that focuses on matrix models. These models could also be applied to the original CPU implementation to parallelize it and speedup its runtime.

4.1 Signal Processing Chain

For our CUDA implementation of the signal processing chain, we focused on empowering the OCT application to leverage optimizations provided by the CUBLAS library. We remodeled the data alignment of the original CPU implementation (see Sect. 3) in matrix notation as shown in Eq. 1. Using the fact, that each A-scan is written continuously into the memory, we used a column-major ordered matrix.

$$\begin{bmatrix} A_1^1 & A_1^2 & \dots & A_1^{d_B} \\ A_2^1 & A_2^2 & \dots & A_2^{d_B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{d_A}^1 & A_{d_A}^2 & \dots & A_{d_A}^{d_B} \end{bmatrix} \tag{1}$$

We exemplify the data interpretation as matrix model by looking in-depth at the white adjust function (`WA` in Fig. 2) which is the hotspot of the CPU implementation. Originally, three consecutive for-loops are used to improve the image quality. The first loop computes the sum, the second one divides by the number of A-scans, and the third loop subtracts the result from the original data. The computation of the average can also be written as a matrix vector product, which can be computed by calling `gemv` provided by the BLAS library. It computes $y \leftarrow \alpha Ax + \beta y$ where A is the obtained B-scan, x is a vector of ones, $\alpha = \frac{1}{\#\text{ A-scans}}$, and $\beta = 0$. The results stored in y are subtracted from each A-scan in A using the `ger` function: $A \leftarrow \alpha y x^T + A$. We assign x as vector of ones, reuse y and A from the previous step and set $\alpha = -1$. Thus, we rewrote three for-loops using two BLAS calls. Using BLAS calls also holds for rewriting `MEAN` and `REF` (see Sect. 3). To map code parts to CUBLAS-specific functions, we also applied matrix models to the functions `PI` and `WIN`, which could be reformulated using `cublasDdgmm` with a weighting vector as diagonal matrix.

For the function `FFT`, we exchanged the FFTW library that is used in the CPU version by NVIDIA's CUFFT library. It performs multiple Fourier Transforms in parallel since the A-scans are independent from each other. For functions which could not be remodeled in matrix notation, we used THRUST whenever possible. It provides a GPU-optimized version of `std::lib` algorithms. With THRUST, self-written transformations can be applied to each value of a data vector. In case of `LOG` and amplitude computations, which is part of `FFT` we used this to perform the needed operations. Finally, we provide self-written kernels for the remaining functions, namely `I2F` and `DC`. An overview of the used principles per function is given in Table 1.

Table 1. Overview of optimization principles for the signal processing functions.

Function	I2F	REF	WA	DC	WIN	FFT	MEAN	LOG	PI
Principle	kernel	CUBLAS	CUBLAS	kernel	CUBLAS	CUFFT+THRUST	CUBLAS	THRUST	CUBLAS

4.2 Data Transfer

To optimize the data transfer to and from the GPU, we keep unmodified data on the GPU and, thus, reduce the amount of data transferred. Data that is modified in each step (B-scans) is copied as a whole to the GPU, resulting in a high bandwidth. Furthermore, we use pinned memory with no ECC for all data transfers as we aimed for asynchronous data transfers. `OCT_sync` is the first code version that the parallelization of all kernels in the signal processing chain while relying on synchronous data transfers.

For the second (further optimized) code version `OCT_async`, we overlap data transfer and compute operations by using CUDA's streaming concept and asynchronous operations. Since the OCT B-scans are independent from each other, each CUDA stream processes one B-scan. Thus, the processing can be executed in one stream and another one moves data at the same time.

5 Performance Model

For validating the performance of the presented CUDA implementation, a performance model which takes the compute and data transfer times with synchronous and asynchronous transfers into account is derived. The results of the performance models also depend on the hardware, as GPU we used a Geforce GTX Titan X and a Geforce 1050 Ti, both of Pascal architecture. An overview of the specifications is given in Table 2.

5.1 Signal Processing

As part of an overall performance model, we take the Boat Hull Model [13] to abstract the OCT signal processing functions on the GPU. The model contains the compute and memory bound of the Roofline Model. As our analysis reveals that all functions are of low computational complexity, we only consider the memory bound. The estimated runtime m_0 for memory bound kernels is given by Eq. 2 with $d = c+u$, where d is the total amount of data accessed, and c and u are the coalesced and uncoalesced memory accesses respectively. The corresponding

Table 2. Specifications of the used GPUs.

	Architecture	Memory	MP	CUDA cores	GPU Clock rate	Mem. Clock rate	Mem. Bus Width
GTX Titan X	Pascal	12 GB	24	3072	1.08 GHz	3505 MHz	384 bit
GTX 1050 Ti	Pascal	4 GB	6	768	1.46 GHz	3504 MHz	128 bit

Table 3. Performance properties of the different used GPUs.

	$P_{coalesced}$	$P_{uncoalesced}$	b_{H2D}	α_{H2D}	b_{D2H}	α_{D2H}
GTX Titan X	230 GB/s	10 GB/s	6.1610 GB/s	0.030 ms	6.7084 GB/s	0.030 ms
GTX 1050 Ti	101 GB/s	12 GB/s	6.1614 GB/s	0.035 ms	6.7066 GB/s	0.050 ms

bandwidths are given by $P_{coalesced}$ and $P_{uncoalesced}$. If only scattered memory accesses occur, m_1 gives the estimated execution time.

$$m_0 = \frac{c}{P_{coalesced}} + \frac{u}{P_{uncoalesced}}, \quad m_1 = \frac{d}{P_{uncoalesced}} \tag{2}$$

To determine the on-device bandwidth we used the SHOC `deviceMemory` benchmark [5]. We used the maximum of the measurements of `readGlobalMemoryCoalesced` as bandwidth for the coalesced memory access $P_{coalesced}$, whereas the lowest value of `readGlobalMemoryUnit` and `writeGlobalMemoryUnit` represents $P_{uncoalesced}$. The latter measures the read or write bandwidth of uncoalesced, per thread contiguous, global memory accesses [5]. Since we optimized the algorithms by using high-performance libraries the particular memory access pattern cannot be reconstructed. Therefore, we needed to base values for c and u in the model on the assumption that the used libraries mainly use contiguous data access. From results of the NVIDIA profiler, we conclude that c is between 60% and 95% of the total data amount d depending on the signal processing function. The GPU-specific model parameters are listed in Table 3.

5.2 Data Transfer

Besides modeling the performance of the single GPU kernels, it is crucial to also incorporate the CPU-GPU data transfer into the performance evaluation for a better representation of the reality. First, we describe a general model for CPU-GPU data transfers where the hardware-dependent parameters are obtained by benchmarks. Later, we modified this model to take the OCT-specific data transfers into account.

For modeling the time of the data transfer, we generally used $T(d) = \alpha + \frac{d}{\beta}$ with data size d in Byte, latency α in seconds, and β the transfer bandwidth [3]. For modeling the data transfer to the GPU, it holds $d = 2B \times d_A \times d_B$, $\alpha = \alpha_{H2D}$, and $\beta = b_{H2D}$. The transfer of the data back to CPU is divided into two parts. First the processed data with $d_A \times d_B$ elements. Secondly, the computed spectrum with $(\frac{d_A}{2} + 1) \times d_B$ elements is copied to the CPU. Each element of the data sets is of type `float`, hence has a size of 4 B. Additionally, the latency α_{D2H} is needed twice, once for each copy operation, as displayed in Eq. 3.

$$T_{D2H}(d_A, d_B) = 2 \cdot \alpha_{D2H} + \frac{4B}{b_{D2H}} \times d_A \times d_B + \frac{4B}{b_{D2H}} \times (\frac{d_A}{2} + 1) \times d_B \quad (3)$$

We used the SHOC benchmarks `BusSpeedDownload` and `BusSpeedReadback` for b_{H2D} and b_{D2H}, and self-written latency benchmarks for α_{H2D} and α_{D2H}. The results are reported in Table 3. Of particular interest are the bandwidths for data sets of 0.5 MB to 64 MB, as our test data sets. Both GPUs reach the maximum of the transfer bandwidth at approx. 30 MB. As we are primarily interested in the performance of high-resolution OCT data sets, the highest attainable bandwidth is used as model parameter. Comparing the model and the results from the benchmark yield a deviation lower than 10%. This deviation occurs mainly at small data sizes since we used the highest attainable bandwidth as model parameter, hence, the transfer times for small data sets are underestimated.

5.3 Synchronous Data Transfer (`OCT_sync`)

The standard copy in CUDA is executed synchronously, meaning that the copy operation first has to be completed before the next processing step can start. For modeling the performance of systems with no overlapping computations or data transfers, the runtime T is the sum of the data transfer time from host to device and vise-versa (T_{H2D} and T_{D2H}) and the runtime of all kernels $T_{proc} = \sum m_0$. Thus, the total processing can be estimated by Eq. 4.

$$T = T_{H2D} + T_{proc} + T_{D2H} \quad (4)$$

5.4 Asynchronous Data Transfer (`OCT_async`)

Modeling data transfer of GPUs with two copy engines (as in the used Pascal GPUs) and no implicit synchronization is objective of [14]. The predicted runtime is the maximum of all possible combinations of overlap as described in Eq. 5. From Sect. 5.1 we concluded that the GPU is not utilized completely since all processing functions are memory bound. Hence, multiple compute operations of different streams can be executed concurrently on the GPU. Thus, the second term in Eq. 5 can be neglected since it describes the case that all computations are executed serially. The first term of Eq. 5 can also be eliminated since the time needed for the copy from device to host is always larger than the transfer from host to device, since more data needs to be transferred. Including the results from the previous steps yields the performance model for a GPU with two copy engines (as in our Pascal GPUs) for the OCT signal processing chain.

$$T = max(T_{H2D} + \frac{T_{proc}}{\#streams} + \cancel{\frac{T_{D2H}}{\#streams}}, \cancel{\frac{T_{H2D}}{\#streams} + T_{proc} + \frac{T_{D2H}}{\#streams}}, \frac{T_{H2D}}{\#streams} + \frac{T_{proc}}{\#streams} + T_{D2H}) \quad (5)$$

6 Results

For evaluating the performance of the OCT signal processing chain, we reduced the software system of Fraunhofer IPT to a test setup focusing on the signal processing functions included as shared libraries in the test suite. These implementations were compiled using either Microsoft's Visual Compiler 14.0 (MS VS), Intel's Compiler 17.0 (ICC) or CUDA 8.0, respectively. Additionally to the provided serial reference implementation (`MSVS`), BLAS libraries are utilized for CPU-parallel versions: `BLAS` (OpenBLAS + MS VS) and `ICC` (MKL + ICC). We used two different test set-ups: First a work station at Fraunhofer IPT, second a compute node of the RWTH Aachen cluster. The work station contains an Intel i7 3820 Sandy Bridge CPU with 3.6 GHz on 4 cores (8 threads) and 16 GB main memory. The compute nodes of the cluster are 2-socket Intel Broadwell EP E5-2650v4@2.2 GHz systems with an overall of 24 cores. Due to unbalanced data affinity, using only one socket with 6 threads and `close` thread binding yields the best-effort performance (`ICC_BW`). Future work will cover improved data affinity.

Runtime measurements on the CPU were tracked using the `boost::timer`, whereas CUDA calls were measured with CUDA events. The measured time also includes some overhead from the program flow of the processing chain. For `OCT_async`, we conducted an overall measurement of 100 runs and then derived the average runtime of a single execution. Furthermore, each of the measurements is the mean of 100 separate runs. We ensured that the standard deviation of measurements is within 10% of the reported mean and the measurements are roughly normally distributed. The number of used streams was set (up) to the number of multiprocessors as this lead to the best results in our tests (see Table 2). Times needed for initial copy operations of constant values to the GPU is not taken into account since it can be neglected when using OCT in real applications.

6.1 Model vs. Measurement

To validate the performance of `OCT_async`, we compare the measured runtimes with our predicted times from the model (see Eq. 5). For the Geforce GTX Titan X, the results are displayed in Fig. 3, tested to a maximum of 16 streams. Comparing `OCT_sync` with the predicted runtimes, shows an error of 5% to 8%. In case of `OCT_async`, the largest error (20%) occurs when using 4 streams, where our implementation has a better performance than predicted by the model. Using more than 4 streams lead to no further performance improvement, contrary more streams tend to lead to a slower processing time for smaller data sets. In case of the Geforce GTX 1050 Ti, the measured and predicted runtimes are shown in Fig. 4. The difference for `OCT_sync` is less than 5% for all tested data sets. Although the Geforce GTX 1050 has 6 multiprocessors, we tested it up to 4 streams, as our tests showed the best performance. The difference between model and reality is up 15%.

For both GPUs, `OCT_sync` is slower as predicted by the model. But when using multiple streams the measured execution time is faster than the model.

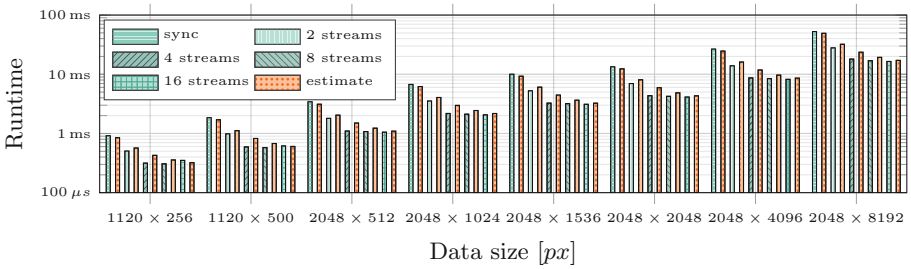

Fig. 3. Comparison of measured and predicted compute times on Geforce GTX Titan X with asynchronous data transfer.

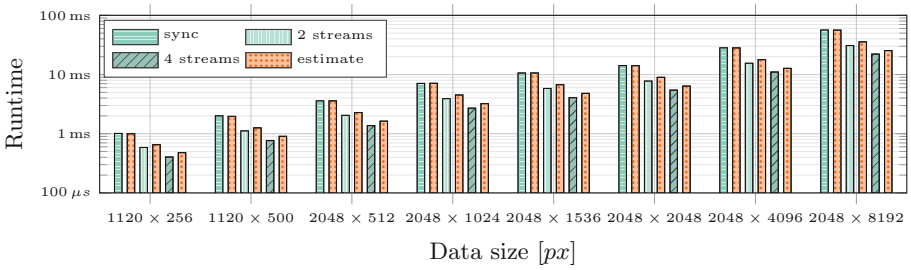

Fig. 4. Comparison of measured and predicted compute times on Geforce GTX 1050 Ti with asynchronous data transfer.

The error is mainly introduced in the prediction of T_{proc}, in particular in the functions where mostly scattered memory accesses occur, i.e. `DC`. The benchmarked $P_{uncoalesced}$ is based on more coalesced memory accesses than the compute kernels, leading to an overestimation of the runtime. However, the performance of our GPU implementations lies close to the predicted runtimes.

6.2 Performance Comparison

With the given (serial) reference implementation (`MSVS`), scans up to a size of 2048×1024 px could be processed with the desired frame rate. In Fig. 5, the processing times of the different implementations and data sizes are shown. Additionally, the speed-up compared to `MSVS` is displayed. The developed CPU parallel versions allow to process our data set with 2048×2048 px in less than 40 ms. They lead to a speed-up between 1.5 and 3, which means up to 3 times higher frame rate compared to the serial implementation. Due to higher clock frequency, we get nearly the same execution time for the Sandy Bridge as for the Broadwell node (`ICC` vs. `ICC_BW`). By using our new implemented GPU version with synchronous data transfer `OCT_sync`, all of the given test data sets could be processed faster than 40 ms with both GPUs. Overall the synchronous GPU

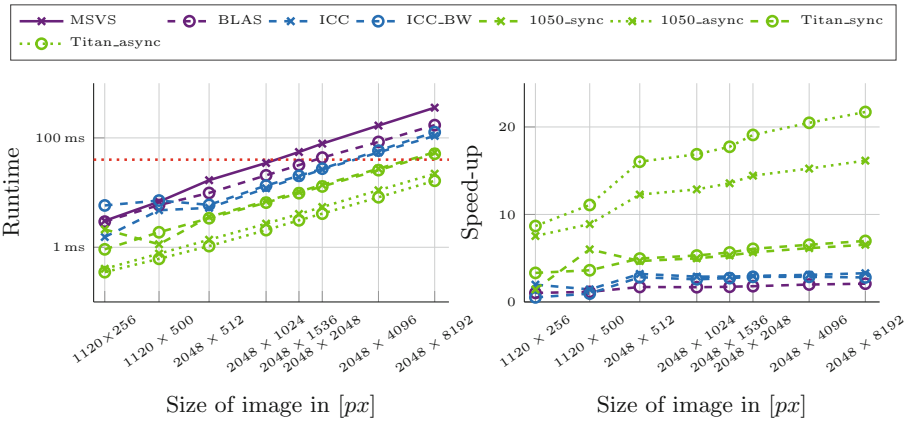

Fig. 5. Runtime comparison (log scale) of the different implementations and Speed-up compared to the serial reference implementation for the OCT signal processing chain.

implementation is 5 to 7 times faster than the provided reference implementation. Compared to the CPU-parallel version a speed-up of 2.5 to 3 is achieved. Now using `OCT_async` reduces the processing time by a factor of 2.5 for the Geforce GTX 1050 Ti compared to `OCT_sync`. Resulting in a speed-up of 8 to 16 compared to `MSVS`, and 4 to 5 compared to the CPU-parallel implementations. For the Geforce GTX Titan X, an additional speed-up of 3 could be noted compared to `OCT_sync`, hence, it can process the OCT signal 8 to 21 times faster than `MSVS`. Compared to the CPU-parallel versions `OCT_async` is 5 to 7 times faster.

7 Conclusion and Outlook

For creating tomographic images with OCT, the signal processing is the limiting factor to achieve a frame rate of 25 frames/s for smoothly displaying the result. In this work, we developed a GPU-parallel version of Fraunhofer IPT's OCT software using CUDA. We further created a corresponding performance model that includes runtime prediction of the OCT kernels and the PCIe data transfers.

For the porting of the signal processing kernels, we focused on re-formulating the algorithms in matrix notation to leverage highly-tuned libraries like CUBLAS. Furthermore, our optimizations included overlapping of data transfers and computations. With our CUDA implementation `OCT_sync`, we achieve a speed-up of factor 5 to 7 on consumer Pascal GPUs over the serial CPU version. Hence, the frame rate is increased from 3 frames/s to 20 frames/s for the largest data set. With `OCT_async`, 45 frames/s are reached for this data set, i.e., a speed-up of 5 compared to the CPU-parallel versions. Our performance model captures all important properties of these OCT GPU implementations. Deviations of measured and modeled performance results are below 15%. Using the

model, we estimate that B-scans up to 2048×24576 px can be processed with video rate.

In future, to achieve further acceleration and enable video frame rates for volumetric 3D-scans, we will constantly optimize the code for both, CPU and GPU. Part of this is using the GPU to directly displaying the obtained signal, hence, saving two copy operations.

References

1. van Aarle, W., et al.: Fast and flexible x-ray tomography using the astra toolbox. Opt. Express **24**(22), 25129–25147 (2016)
2. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: LogGP: incorporating long messages into the logp model for parallel computation. J. Parallel Distrib. Comput. **44**(1), 71–79 (1997)
3. Boyer, M., Meng, J., Kumaran, K.: Improving GPU performance prediction with data transfer modeling. In: 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum, pp. 1097–1106, May 2013
4. Culler, D.E., et al.: LogP: a practical model of parallel computation. Commun. ACM **39**(11), 78–85 (1996)
5. Danalis, A., et al.: The scalable heterogeneous computing (SHOC) benchmark suite. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, GPGPU-3, pp. 63–74. ACM, New York (2010)
6. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proc. IEEE **93**(2), 216–231 (2005). special issue on "Program Generation, Optimization, and Platform Adaptation"
7. Drexler, W., Fujimoto, J.G.: Optical Coherence Tomography: Technology and Applications. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77550-8
8. Gómez-Luna, J., González-Linares, J.M., Benavides, J.I., Guil, N.: Performance models for asynchronous data transfers on consumer graphics processing units. J. Parallel Distrib. Comput. **72**(9), 1117–1126 (2012). accelerators for High-Performance Computing
9. Huang, D., et al.: Optical coherence tomography. Science **254**(5035), 1178–1181 (1991)
10. Izatt, J.A., Choma, M.A.: Theory of optical coherence tomography. In: Drexler, W., Fujimoto, J.G. (eds.) Optical Coherence Tomography. Biological and Medical Physics, Biomedical Engineering, pp. 47–72. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77550-8_2
11. Van der Jeught, S., Bradu, A., Podoleanu, A.G.: Real-time resampling in fourier domain optical coherence tomography using a graphics processing unit. J. Biomed. Opt. **15**(3), 030511–030511–3 (2010)
12. Madougou, S., Varbanescu, A., de Laat, C., van Nieuwpoort, R.: The landscape of GPGPU performance modeling tools. Parallel Comput. **56**, 18–33 (2016)
13. Nugteren, C., Corporaal, H.: The boat hull model: enabling performance prediction for parallel computing prior to code development. In: Proceedings of the 9th Conference on Computing Frontiers, CF 2012. ACM Press (2012)
14. Van Werkhoven, B., Maassen, J., Seinstra, F.J., Bal, H.E.: Performance models for CPU-GPU data transfers. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 11–20, May 2014

15. Wieser, W., Draxinger, W., Klein, T., Karpf, S., Pfeiffer, T., Huber, R.: High definition live 3D-OCT in vivo: design and evaluation of a 4D OCT engine with 1 GVoxel/s. Biomed. Opt. Express **5**(9), 2963–2977 (2014)
16. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009)
17. Zhang, K., Kang, J.U.: Graphics processing unit-based ultrahigh speed real-time fourier domain optical coherence tomography. IEEE J. Sel. Top. Quantum Electron. **18**(4), 1270–1279 (2012)
18. Zhang, K., Kang, J.U.: Real-time 4D signal processing and visualization using graphics processing unit on a regular nonlinear-k fourier-domain oct system. Opt. Express **18**(11), 11772–11784 (2010)