GPU Optimization of Convolution for Large 3-D Real Images

Pavel Karas[1], David Svoboda[1], and Pavel Zemčík[2]

[1] Centre for Biomedical Image Analysis, Faculty of Informatics, Masaryk University, Botanická 68a, Brno, CZ
{xkaras1,svoboda}@fi.muni.cz
[2] Dept. of Computer Graphics and Multimedia, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, Brno, CZ
zemcik@fit.vutbr.cz

Abstract. In this paper, we propose a method for computing convolution of large 3-D images with respect to real signals. The convolution is performed in a frequency domain using a convolution theorem. Due to properties of real signals, the algorithm can be optimized so that both time and the memory consumption are halved when compared to complex signals of the same size. Convolution is decomposed in a frequency domain using the decimation in frequency (DIF) algorithm. The algorithm is accelerated on a graphics hardware by means of the CUDA parallel computing model, achieving up to 10× speedup with a single GPU over an optimized implementation on a quad-core CPU.

1 Introduction

Convolution is one of the essential operations in both image and signal processing. In some applications, it can be also viewed as a *filter* of a signal f, parametrized by g, so-called *filter kernel*. A kernel is given by so-called point spread function (PSF), a function that describes the impulse response of an imaging system to a point source [14]. The PSF can be a simple function; for example, Gaussian is a typical representative of common convolution kernel. In some applications, such as optical microscopy, the PSF can be a non-analytic function, obtained e.g. from empirical measurements of an optical system. An example of the convolution is shown in Fig. 1.

The convolution can be employed for blurring images, edge detection, noise suppression, image registration, and in many other applications [8,14,26]. It can be used to simulate image formation in optical systems, such as optical microscopes [32], as shown in Fig. 1. In image restoration, it is employed as an essential part of deconvolution algorithms [24,29,34].

The convolution can be time-demanding. A naïve convolution of two discrete signals f and g, according to the definition, has the computational complexity of $\mathcal{O}(M_f M_g)$ where M_f and M_g are number of samples of f and g, respectively. In some applications, this is sufficient since the kernel is usually small (hundreds or

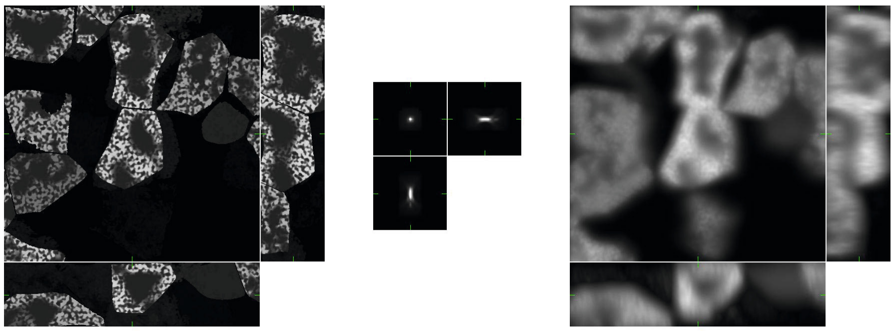

(a) Phantom image (b) PSF (c) Blurred image

Fig. 1. Example of a 3-D convolution. The images show an artificial (phantom) image of a tissue, a PSF of an optical microscope, and blurred image, computed by the convolution of the two images. Each 3-D image is represented by three 2-D views (XY, YZ, and XZ).

thousands of samples in maximum) or it is separable [2]. In other applications—such as optical microscopy—one deals with millions of samples. In this case, it is advisable to compute the convolution in the frequency domain, according to the so-called convolution theorem [2]. This approach allows to decrease the computational complexity to $\mathcal{O}(M \log M)$ where $M = M_f + M_g$ [35]. In practice, this means that the computation can take seconds or minutes instead of hours or days. The comparison of convolution in the spatial and the frequency domain was made e.g. in [5]. Further speed-up can be achieved using graphics cards.

At present, graphics processing units (GPU) are used not only for visualisation purposes but also to accelerate general computations. This phenomenon is often referred to as general-purpose computing on graphics processing units (GPGPU) [23]. Recently, two programming frameworks are widely used among the GPGPU community, namely CUDA [21] and OpenCL [10].

The GPU implementations of a naïve convolution and a convolution with separable kernel can be found in [25]. These algorithms can be used in many applications, such as fast computation of Canny edge detection [16,22]. As for the convolution in the frequency domain, the essential part of this approach is the Fourier transform. Recently, the CUFFT library [19] by NVIDIA offers a framework for implementing convolution in a straightforward manner. Besides CUFFT, other FFT libraries for GPU were developed, such as [9,18]. GPU acceleration of FFT and convolution in practical applications have been well described in literature [4,5].

Relatively small global memory of the GPU architecture poses a significant problem in applications where one deals with huge images. Several approaches exist to decompose the convolution into sub-problems, such as the partition in the spatial domain described in [33] and succesfully adopted in [1,31]. In [15], authors proposed a new method, based on the decimation in frequency (DIF) algorithm and specifically designed for the GPU architecture. This approach allows efficient computation of the convolution on GPU that is not limited by the size of the GPU memory. The main drawback is that it is designed for complex input data, hence it is relatively inefficient when processing real signals.

In this paper, we propose an optimized method for convolution of huge images on GPU which consists of two concepts: (i) the decomposition of the problem, as proposed in [15], (ii) the optimization for real input data (i.e. data with zero imaginary part) which are of interest in most practical applications. In Section 2, we recall the convolution, some of its properties, and the decomposition concept. In Section 3, we recall approaches to optimize the convolution for real data. We consider which one is the most suitable to be combined with the decomposition. In Section 4, we test the performance and the precision of the GPU implementation. Finally, the conclusions summarize the main contributions of our work.

2 Convolution

A 1-D convolution of two discrete finite signals f, g is defined by following:

$$[f * g](m') \equiv \sum_{m=0}^{M_g-1} f(m'-m)g(m), \quad m' = 0, \ldots, M-1, \tag{1}$$

where M_f and M_g is the number of samples of f and g, respectively. The convolution then produces a signal of size $M = M_f + M_g - 1$. The convolution can be extended to any number of dimensions. For details, refer to [2,11].

2.1 Convolution Theorem

An efficient approach to compute a convolution is given by so-called convolution theorem. Having two periodic signals f, g, it can be proved that

$$f * g = \mathcal{F}^{-1}\left[\mathcal{F}[f]\mathcal{F}[g]\right], \tag{2}$$

where $\mathcal{F}$ denotes a discrete Fourier transform (DFT).

2.2 Decimation in Frequency (DIF) Algorithm

FFT algorithms are based on the divide-and-conquer approach. To perform the data division, two algorithms can be used: decimation in time (DIT) and decimation in frequency (DIF) [3]. We will introduce the idea of the DIF algorithm for the 1-D case. Let us have a function $f(m)$ and its Fourier transform $F(\mu)$, $m, \mu = 0, \ldots, M-1$. Supposing that M is even we introduce new functions $r(m')$ and $s(m')$, $m' = 0, \ldots, M/2-1$ as follows [28,12]:

$$r(m') \equiv f(m') + f(m' + M/2), \tag{3a}$$

$$s(m') \equiv \left[f(m') - f(m' + M/2)\right] W_M^{-m'}, \tag{3b}$$

where $W_M = \mathrm{e}^{\mathrm{i}\frac{2\pi}{M}}$. Vice versa, it is simple to deduce

$$f(m') = \left[r(m') + s(m')W_M^{m'}\right]/2, \tag{4a}$$

$$f(m' + M/2) = \left[r(m') - s(m')W_M^{m'}\right]/2. \tag{4b}$$

Then it can be proved that the Fourier transforms $R(\mu')$ and $S(\mu')$ of the functions $r(m')$ and $s(m')$ fulfil the following property:

$$R(\mu') = F(2\mu'), \tag{5a}$$

$$S(\mu') = F(2\mu' + 1). \tag{5b}$$

2.3 GPU Accelerated Convolution

In [15], authors proposed the efficient GPU implementation of convolution of large 3-D images. The algorithm has three phases: (i) both signal and kernel are decomposed into $\mathcal{P}$ parts on CPU, using the DIF algorithm; (ii) the convolution is computed piecewise in the frequency domain on GPU; (iii) the result is composed from the subparts on CPU. The scheme of the algorithm is shown in Fig. 2.

The most important contribution of this approach is that the computation is not limited by the size of GPU memory which is usually significantly smaller than CPU memory. The main drawback is that the algorithm is designed for complex input data; therefore, it is sub-optimal in most applications. In this paper, we will describe how this algorithm can be further optimized for real input data. We will show that a significant improvement can be achieved in means of both the time and the memory complexity.

The method specified in the following section consists basically of two concepts. One, already described in [15], will be referred to as *data decomposition.* The new concept, introduced in the section 3.1, will be reffered to as *real data optimization.*

3 Method

3.1 Fourier Transform of a Real Signal

The (discrete) Fourier transform of a real signal keeps some specific properties, which can be used for further optimization, when processing real images. Many of these properties were described in the literature [2,13,27]. In particular, if the input signal $f(m)$, $m = 0, 1, \ldots, M-1$ is real, then the following property is held:

$$F(m) = F^*(M - m). \tag{6}$$

As a result, one half of the output data is redundant. It is reasonable not to compute redundant data in order to reduce computation complexity as well as memory requirements.

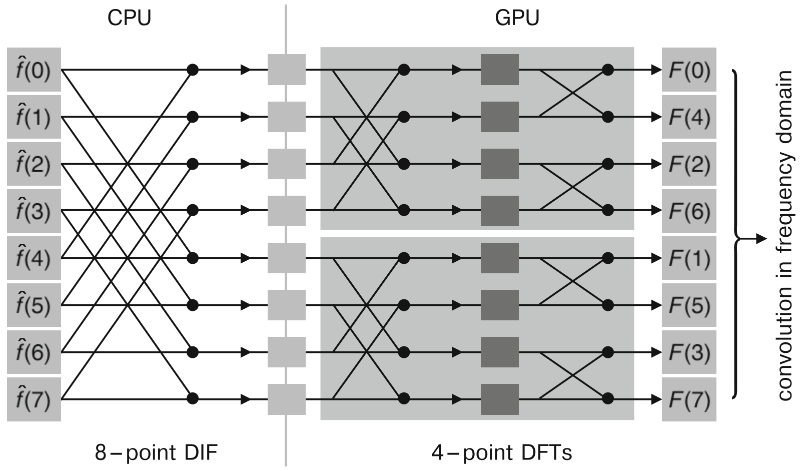


Fig. 2. A scheme description of the convolution algorithm with a decomposition in a frequency domain proposed in [15]. An input signal is decomposed into 2 parts by the decimation in frequency (DIF) algorithms, i.e. $\mathcal{P} = 2$. The parts are subsequently processed independently on GPU, using the discrete Fourier transform (DFT). Moving the border line between CPU and GPU to the right is equivalent to setting $\mathcal{P} = 4, 8, \ldots$

When optimizing computation of DFT of the aforementioned input signal $f(m)$, several approaches can be considered. In the first—used by most popular implementations of DFT, including the FFTW [7] and the CUFFT [19] libraries—only the half transform $F(m')$, $m' = 0, 1, \ldots, M/2$ is computed. Yet it is difficult to combine this approach with our decomposition method. Firstly, it requires padding the input signal (for details, refer to [7]) that leads to reallocating of the whole memory block. Secondly, the "R2C"[1] method implemented in CUFFT, according to our experiments, is less efficient than a "classic C2C"[2] method applied to a complex signal of half size.

In the second approach, two real input signals $f(m)$, $g(m)$ of the same size are combined into one complex signal $h(m) = f(m) + \mathrm{i}g(m)$ of the same length. Unfortunately, this combination requires creating an additional buffer of at least the size of f. This places higher demands on CPU memory.

Our method is based on the third approach which uses the following idea: A real input signal $f(m)$, $m = 0, \ldots, M-1$ is processed as a complex signal $\hat{f}(m')$, $m' = 0, \ldots, M/2 - 1$ of the half size (provided that the size of the input signal is even):

$$\hat{f}(m') \equiv f(2m') + \mathrm{i}f(2m' + 1). \tag{7}$$

Using the common representation of real and complex numbers in the C language, this means that a real signal can be turned into a complex one by simply over-casting the data type, avoiding any data transfers. The relationship between

[1] Real-to-complex.

[2] Complex-to-complex.

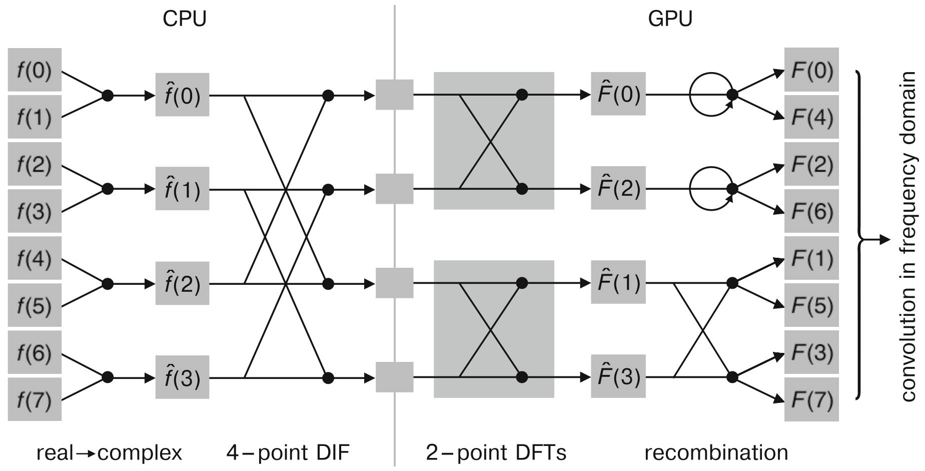


Fig. 3. A scheme description of the modified algorithm, $\mathcal{P} = 2$. For a comparison with original algorithm, refer to Fig. 2.

the Fourier transform $F(m)$ of the real signal $f(m)$ and the Fourier transform $\hat{F}(m')$ of the complex signal $\hat{f}(m')$ is given by following [12]:

$$F(m') = \frac{1}{2}\left(\alpha_+(m') - \mathrm{i}W_M^{-m'}\alpha_-(m')\right), \tag{8a}$$

$$F(m' + M/2) = \frac{1}{2}\left(\alpha_+(m') + \mathrm{i}W_M^{-m'}\alpha_-(m')\right), \tag{8b}$$

where

$$\alpha_\pm(m') \equiv \hat{F}(m') \pm \hat{F}^*(M/2 - m'). \tag{9}$$

3.2 Optimization of the Decomposition Algorithm

The real data optimization, described above, can be seamlessly combined with the data decomposition method, proposed in [15]. The optimization basically requires no modifications to the composition and decomposition functions. Some changes must be made to the CUDA kernel which computes the point-wise multiplication so that it incorporates the recombination described in Eq. (8a) and (8b). The scheme of the modified algorithm is shown in Fig. 3.

It can be noted that the algorithms for both decimation and array permutation were originally introduced in 1970s not only to reduce the time complexity of FT but also to allow efficient use of memory spaces available [6]. In this context, our approach can be viewed as a revival of such techniques.

3.3 Getting Further

One has to take into account more dimensions when processing d-dimensional data. For implementation reasons, described in the previous section, it is reasonable to perform real data optimization in the last (usually x) axis. On the other hand, to achieve maximum data transfer efficiency, it is advisable to perform the decomposition in the first (y for $d = 2$ or z for $d = 3$) axis, as explained in [15].

For example, in 2-D space, Eq. (8a) and (8b) can be extended as follows:

$$F(m', n) = \frac{1}{2}\left(\alpha_+(m', n) - \mathrm{i}W_M^{-m'}\alpha_-(m', n)\right), \tag{10a}$$

$$F(m' + M/2, n) = \frac{1}{2}\left(\alpha_+(m', n) + \mathrm{i}W_M^{-m'}\alpha_-(m', n)\right), \tag{10b}$$

where $m' = 0, 1, \ldots, M/2 - 1$, $n = 0, 1, \ldots, N - 1$, and

$$\alpha_\pm(m', n) \equiv \hat{F}(m', n) \pm \hat{F}^*(M/2 - m', N - n). \tag{11}$$

The real data optimization can be combined with data decomposition for any number of parts $\mathcal{P}$ that the data are decomposed into. It should be noted that due to the recombination phase, memory requirements change for $\mathcal{P} > 2$. Whereas first two parts recombine with themselves, the others recombine in pairs. Therefore, the GPU memory requirements are no longer $(M_f + M_g)/\mathcal{P}$ but $2(M_f + M_g)/\mathcal{P}$. For a practical example, refer to the following section.

4 Experimental Results

Experiments were conducted on a machine described in Table 1. The CPU implementation uses the multi-threaded FFTW library while the GPU implementation uses our algorithm along with the CUFFT library. The decomposition and the composition functions are performed on CPU and improved with SSE intrinsics for a better performance.

Table 1. Machine used for experiments

CPU/GPU	# of cores	Clock speed	RAM size	Bandwidth
Intel Core i7 950	4	3.07 GHz	6 GB	12.8 GB/s
NVIDIA GeForce GTX 480	480	1.40 GHz	1.5 GB	88.7 GB/s

In the experiments, we used randomly generated images of certain sizes. Two datasets were created. In the first dataset, the image dimensions were powers of 2 which allows most efficient computation of FFT. In the second dataset, the images were arbitrarily sized except that in the z dimension the padded size is kept so that the decomposition can be performed without the need of additional image padding. We refer to the images from the two datasets as *specifically* and *arbitrarily* sized, respectively. For details, refer to Table 2.

In the first experiment, we compared CPU and GPU implementations for a single value of $\mathcal{P}$, various image sizes, and both real and complex input data. The parameter $\mathcal{P}$ was chosen the smallest possible in all cases, allowing the best GPU performance. As shown in the following experiment, and described in [15], the

Table 2. Sizes of the images used in experiments

Image size [Mpx]	1st dataset	2nd dataset
34	$512 \times 512 \times 128$	$514 \times 514 \times 128$
67	$1024 \times 512 \times 128$	$1028 \times 514 \times 128$
134	$1024 \times 1024 \times 128$	$1028 \times 1028 \times 128$
268	$1024 \times 1024 \times 256$	$1028 \times 1028 \times 256$

GPU performance slightly decreases when increasing $\mathcal{P}$. The results are shown in Fig. 4(a),(b). For better visual comparison of implementations in plots, the performance P is computed as $P = s/t$, where s is the output image size and t is the computation time including data transfers between CPU and GPU.

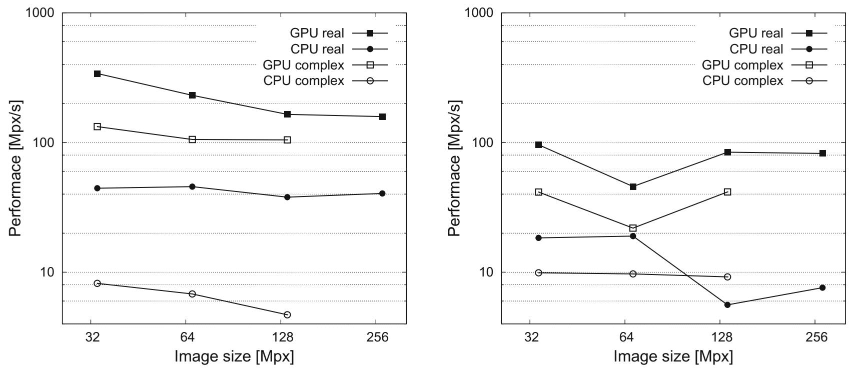
The results clearly show that the real data optimization not only increases the performance significantly, it also allows processing images that could not be processed without the optimization, due to insufficient CPU memory. The GPU performance slightly decreases as the image size increases. This is in fact due to increase of the $\mathcal{P}$ parameter. Still, the speed-up over the CPU implementation is approx. $4\times$ for the 1st dataset and up to $10\times$ for the 2nd dataset.

In the second experiment, we measured the performance of CPU and GPU implementations for various values of $\mathcal{P}$, fixed image size, and real input data. We also evaluated the GPU memory requirements. The results are shown in Fig. 5(a),(b). The experiment confirmed that the GPU performance decreases with the increase of the $\mathcal{P}$ parameter. On the other hand, the amount of the GPU memory required decreases except the step between $\mathcal{P} = 2$ and $\mathcal{P} = 4$. This is due to algorithm properties, explained in Section 3.3.

4.1 Multi-GPU Performance

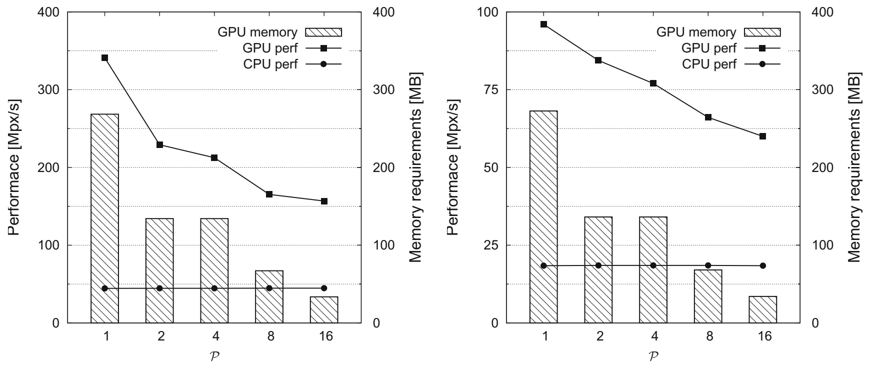
As described in [15], the decomposition algorithm can be adopted for multi-GPU systems, allowing further speed-up. However, bandwidth of data transfers is limited by the PCI-Express bus data processing [30]. This has negative impact on GPU performance and can cause even its decrease. In the third experiment, we measured performance of 2 GPUs computing simultaneously, using the same system with an additional graphics card, namely the NVIDIA GeForce GTX 285. Again, we measured the performance for various values of $\mathcal{P}$ and fixed image size.

The results, shown in Table 3, indicate that in the case of a specifically-sized image (columns 2 and 3), the data transfer overhead is crucial. Hence, adding the second GPU brings no speed-up. In case of an arbitrarily-sized image (columns 3 and 4), the computation of FFT is the most time-demanding phase of the algorithm so with 2 GPUs, a significant speed-up can be achieved. In general, the contribution of the multi-GPU implementation increases with the increase of $\mathcal{P}$.



(a) 1st dataset (specifically sized images) (b) 2nd dataset (arbitrarily sized images)

Fig. 4. Comparison of CPU and GPU implementations for a single value of $\mathcal{P}$ (best choice), various image sizes, and both real and complex input data



(a) $512 \times 512 \times 128$ (b) $514 \times 514 \times 128$

Fig. 5. Comparison of CPU and GPU implementations for various values of $\mathcal{P}$, fixed image size, and real input data. Plots also show GPU memory requirements.

Table 3. Computing time of the GPU implementation: 1 GPU vs 2 GPUs

Image size	$512 \times 512 \times 128$		$514 \times 514 \times 128$	
$\mathcal{P}$	1 GPU [ms]	2 GPUs [ms]	1 GPU [ms]	2 GPUs [ms]
2	146.4	176.7	403.7	293.6
4	157.9	175.8	442.5	274.6
8	203.0	192.4	515.9	275.9
16	213.9	201.2	568.4	298.6

4.2 Precision Analysis

In this experiment, we evaluated the error of the output data in a practical application. The tissue image from Fig. 1(a) was cropped to various sizes and convolved with the PSF from Fig. 1(b) of size $128 \times 128 \times 96$. The input data was 16-bit integers. The convolution was computed on GPU in the single precision and on CPU in the double precision. The error $\delta_{\max}$ was computed as a maximum difference between the two output images. The results for various output image sizes and various values of $\mathcal{P}$ are shown in Table 4. In some cases, the convolution could not be computed for $\mathcal{P} = 1$, due to insufficient GPU memory.

Table 4. Computation error

Image size	$\delta_{\max}$				
$x \times y \times z$	$\mathcal{P} = 1$	$\mathcal{P} = 2$	$\mathcal{P} = 4$	$\mathcal{P} = 8$	$\mathcal{P} = 16$
$384 \times 384 \times 160$	0.006	0.097	0.112	0.105	0.105
$640 \times 384 \times 160$	0.007	0.143	0.112	0.117	0.122
$640 \times 640 \times 160$	0.008	0.225	0.211	0.166	0.121
$640 \times 640 \times 224$	0.009	0.177	0.162	0.132	0.103
$386 \times 386 \times 160$	0.018	0.100	0.113	0.106	0.107
$642 \times 386 \times 160$	0.017	0.146	0.112	0.122	0.125
$642 \times 642 \times 160$	—	0.226	0.213	0.169	0.125
$642 \times 642 \times 224$	—	0.178	0.164	0.133	0.113

The results show that the error is significantly smaller for $\mathcal{P} = 1$, i.e. when no decomposition is performed. Nevertheless, the error does not grow with the increase of $\mathcal{P}$. As $\delta_{\max} < 1$ in all cases, the output 16-bit integer images are virtually the same. Still, it is likely that in some practical applications, the single precision will not be enough. Fortunately, the recent GPU architecture allows efficient computation in the double precision [17,20].

5 Conclusions

We have reviewed efficient approaches to compute convolution of large 3-D images, with special regard to the currently popular GPU architecture. At present, the GPU architecture achieves significantly higher performance than the common multi-core CPU architecture. The main drawback of GPU is relatively small memory which can impose limitations on practical applications, such as optical microscopy, where huge images are of interest. In this paper, we have recalled our GPU implementation of convolution which allows to decompose the problem into sub-parts in an efficient way.

The main contribution of this paper is the optimization of the aforementioned method for the real data which are of interest in most practical applications. Thanks to the optimization, the GPU implementation achieves up to 10× speed-up—including data transfers—over the multi-core CPU implementation, namely the one in the FFTW library which is widely used and considered the state of the art. The GPU implementation is able to compute the convolution of two 270 Mpx images within less than 2 s.

Our experiments proved the GPU implementation to be not only efficient but also precise enough for the optical microscopy images. In applications where the precision is crucial, double precision can be used.

Both concepts, the decomposition and the optimization, are designed to be performed in-place in the CPU memory which allows the maximum exploitation of resources. However, in some applications, the images can even exceed the CPU memory. Here, our method can potentially be combined with the approach described in [31] so that the convolution can be decomposed on multiple levels. Furthermore, the proposed method is not strictly dependent on CUDA nor on the GPU architecture. It can be implemented also in OpenCL and other languages. Other parallel architectures can be taken into account as well. Generally, it can be implemented on a heterogeneous cluster of computers allowing both CPU and GPU to take part in the computation.

Acknowledgements. This work has been supported by the Ministry of Education of the Czech Republic (Project No. 2B06052) and the Grant Agency of the Czech Republic (Grant No. P302/12/G157).

References

1. Boden, A.F., Redding, D.C., Hanisch, R.J., Mo, J.: Massively parallel spatially variant maximum-likelihood restoration of Hubble Space Telescope imagery. J. Opt. Soc. Am. A 13(7), 1537–1545 (1996)
2. Bracewell, R.N.: The Fourier Transform and Its Applications, 3rd edn. McGraw-Hill (2000)
3. Brigham, E.: Fast Fourier Transform and Its Applications, 1st edn. Prentice-Hall (1988)
4. Domanski, L., Vallotton, P., Wang, D.: Two and Three-Dimensional Image Deconvolution on Graphics Hardware. In: Proceedings of the 18th World IMACS/MODSIM Congress, Cairns, Australia, July 13-17, pp. 1010–1016 (2009)
5. Fialka, O., Cadik, M.: FFT and Convolution Performance in Image Filtering on GPU. In: Tenth International Conference on Information Visualization, IV 2006, pp. 609–614 (2006)
6. Fraser, D.: Array permutation by index-digit permutation. J. ACM 23(2), 298–309 (1976), `http://doi.acm.org/10.1145/321941.321949`
7. Frigo, M., Johnson, S.G.: FFTW 3.2.2. Massachusetts Institute of Technology (July 2009), `http://www.fftw.org/fftw3.pdf`
8. Gonzales, R.C., Woods, R.E.: Digital Image Processing, 2nd edn. Prentice-Hall (2002)

9. Govindaraju, N.K., Lloyd, B., Dotsenko, Y., Smith, B., Manferdelli, J.: High performance discrete Fourier transforms on graphics processors. In: SC 2008: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 1–12. IEEE Press, Piscataway (2008)
10. Group, K.: OpenCL (2011), `http://www.khronos.org/opencl/`
11. Hanna, J.R., Rowland, J.H.: Fourier Series, Transforms, and Boundary Value Problems, 2nd edn. John Wiley & Sons (1990)
12. Hey, A.: The FFT Demystified. Engineering Productivity Tools Ltd., 21 Leaveden Road, Watford, Hertfordshire, UK (1999), `http://www.engineeringproductivitytools.com/stuff/T0001/PT10.HTM`
13. Ifeachor, E.C., Jervis, B.W.: Digital Signal Processing: A Practical Approach, 2nd edn. Pearson Education (2002)
14. Jähne, B.: Digital Image Processing, 6th edn. Springer (2005)
15. Karas, P., Svoboda, D.: Convolution of large 3D images on GPU and its decomposition. EURASIP Journal on Advances in Signal Processing (120), 1–12 (2011), `http://asp.eurasipjournals.com/content/2011/1/120`
16. Luo, Y., Duraiswami, R.: Canny edge detection on NVIDIA CUDA. In: Computer Vision and Pattern Recognition Workshop, pp. 1–8 (2008)
17. Nickolls, J., Dally, W.: The GPU Computing Era. IEEE Micro 30, 56–69 (2010), `http://dx.doi.org/10.1109/MM.2010.41`
18. Nukada, A., Ogata, Y., Endo, T., Matsuoka, S.: Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In: SC 2008: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 1–11. IEEE Press, Piscataway (2008)
19. NVIDIA Corporation: CUDA$^{\text{TM}}$ CUFFT Library 2.3 (June 2009), `http://developer.nvidia.com/object/cuda_2_3_downloads.html`
20. NVIDIA Corporation: FERMI Tuning Guide (August 2010), `http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/Fermi_Tuning_Guide.pdf`
21. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, USA: NVIDIA GPU Computing Developer Home Page (June 2011), `http://developer.nvidia.com/category/zone/cuda-zone`
22. Ogawa, K., Ito, Y., Nakano, K.: Efficient canny edge detection using a GPU. In: International Conference on Natural Computation, pp. 279–280 (2010)
23. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware, pp. 21–51 (August 2005)
24. Pankajakshan, P.: Blind Deconvolution for Confocal Laser Scanning Microscopy. Ph.D. thesis, Universite de Nice Sophia Antipolis (December 2009), `http://tel.archives-ouvertes.fr/tel-00474264/fr/`
25. Podlozhnyuk, V.: Image Convolution with CUDA (June 2007), `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_64_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf`
26. Pratt, W.K.: Digital Image Processing, 3rd edn. John Wiley & Sons (2001)
27. Rabiner, L.R.: On the use of symmetry in fft computation. IEEE Transactions on Acoustics, Speech, and Signal Processing 27, 233–239 (1979)
28. Saidi, A.: Generalized FFT Algorithm. In: IEEE International Conference on Communications 93: Technical program, conference record. In: IEEE International Conference on Communications, Geneva, Switzerland, May 23-26, vols. 1-3, pp. 227–231 (1993)
29. Sarder, P., Nehorai, A.: Deconvolution methods for 3-D fluorescence microscopy images. IEEE Signal Processing Magazine 23(3), 32–45 (2006)

30. Schaa, D., Kaeli, D.: Exploring the multiple-GPU design space. In: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, pp. 1–12. IEEE Computer Society, Washington, DC (2009)
31. Svoboda, D.: Efficient Computation of Convolution of Huge Images. In: Maino, G., Foresti, G.L. (eds.) ICIAP 2011, Part I. LNCS, vol. 6978, pp. 453–462. Springer, Heidelberg (2011)
32. Svoboda, D., Kozubek, M., Stejskal, S.: Generation of Digital Phantoms of Cell Nuclei and Simulation of Image Formation in 3D Image Cytometry. Cytometry Part A 75A(6), 494–509 (2009)
33. Trussell, H., Hunt, B.: Image restoration of space variant blurs by sectioned methods. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 1978, vol. 3, pp. 196–198 (1978)
34. Verveer, P.J.: Computational and optical methods for improving resolution and signal quality in fluorescence microscopy. Ph.D. thesis, Technische Universiteit Te Delft (1998)
35. Press, W.H., Teukolsky, S.A., Vettrling, W.T., Flannery, B.P.: Numerical Recipes in C, 2nd edn., ch. 7. Cambridge University Press (1992)