Acceleration of the Split-Step Fourier Method by using a Graphics Processing Unit (GPU)

Technical Report

September 20, 2009

Technische Universität München
Institute for Communications Engineering

http://www.lnt.ei.tum.de

http://www.tum.de

Acceleration of the Split-Step Fourier Method by using a Graphics Processing Unit (GPU)

Technical Report

Institute for Communications Engineering
Technische Universität München

Stephan Hellerbrand* and Norbert Hanik

September 20, 2009

1 Introduction

The simulation of optical transmission systems plays an important role both in research and network-design. Solving the nonlinear Schrödinger equation, which describes the signal evolution in an optical fiber, is usually the most time-consuming task. This holds in particular for long test signals, which have to be used to account for the channel memory in high-speed transmission systems [WEG$^+$04]. Therefore the optimization of optical communication links can take very long even if effective optimization algorithms are used [CGH09].

In addition to that, in modern phase modulated and polarization multiplexed systems the nonlinear interaction of signal and noise as well as polarization mode dispersion (PMD) can often not be neglected. For systems, in which semi-analytical models such as [CMG$^+$09] are not applicable Monte-Carlo sampling has to be used. The corresponding large number of evaluations of the signal propagation can lead to a very large time consumption.

In this article we describe how the computation of the signal propagation in an optical fiber can be significantly accelerated by using graphics processing units (GPU). This is achieved by decomposing the Split-Step Fourier Method (SSFM) into parallel operations and distributing those operations onto the massively parallel graphics hardware. Here we concentrate on commodity graphics hardware, which can be programmed via the NVIDIA Compute Unified Device Architecture (CUDA)[Hal08].

This paper is organized as follows. First we will review the SSFM, upon which our fiber simulation is based. Subsequently we will describe the implementation using a GPU. After that simulation results will be presented, which indicate a significant acceleration of simulations if the GPU is employed instead of a state-of-the art central processing unit (CPU). In addition, the accuracy of the simulations will be discussed and compared to simulations performed using the CPU only.

*Corresponding author is Stephan Hellerbrand, stephan.hellerbrand@mytum.de.

2 Simulation of Signal Propagation on Optical Fiber

The propagation of light in an optical single-mode fiber is governed by the nonlinear Schrödinger Equation. If the propagation is invariant with respect to light polarization then the scalar signal Nonlinear Schrödinger Equation is sufficient for modeling signal propagation:

$$\frac{\partial}{\partial z} u\left(z,t\right) = \left(\mathfrak{d}(t) + \mathfrak{n}(z,t)\right) u\left(z,t\right), \tag{1}$$

where the linear- and nonlinear operators are defined as

$$\mathfrak{d}(t) = \left(j\frac{\beta_2}{2}\frac{\partial^2}{\partial t^2} + \frac{\beta_3}{6}\frac{\partial^3}{\partial t^3} - \frac{\alpha}{2}\right) \quad \text{and} \quad \mathfrak{n}(z,t) = -j\gamma\left|u\left(z,t\right)\right|^2, \tag{2}$$

respectively, and β_2 and β_3 are the 2nd and 3rd order dispersion parameters, α reflects the attenuation of the fiber and γ is the nonlinear coefficient. In this description the time frame is assumed to be moving at group velocity β_1, or $t = t' - \beta_1 z$, where t' is the natural time.
Formally the solution can be written as

$$u\left(z+\Delta z,t\right) = \exp\left(\left(\mathfrak{d}(t) + \mathfrak{n}(z,t)\right)\Delta z\right) \cdot u\left(z,t\right), \tag{3}$$

which reflects the fact that dispersion and nonlinearity act upon the signal simultaneously. Mathematically this interplay manifests itself as two non commuting operators. However, if one considers a very small step Δz the two effects lend themselves to separate evaluation and one obtains an approximate result:

$$u\left(z+\Delta z,t\right) \approx \exp\left(\mathfrak{d}(t)\Delta z\right) \cdot \exp\left(\mathfrak{n}(z,t)\Delta z\right) \cdot u\left(z,t\right). \tag{4}$$

This is schematically depicted in Fig. 1a). The error, which is incurred by this operation can be quantified using the Zassenhaus-expansion of the Baker-Haussdorf formula [WM62, Mag54] for non commuting operators $\mathfrak{a}$ and $\mathfrak{b}$, which is given by

$$\exp\left(\mathfrak{a}\right) \cdot \exp\left(\mathfrak{b}\right) = \exp\Big(\mathfrak{a} + \mathfrak{b} + \underbrace{\frac{1}{2}\left[\mathfrak{a},\mathfrak{b}\right] + \frac{1}{3}\left[\left[\mathfrak{a},\mathfrak{b}\right],\mathfrak{b}\right] + \frac{1}{6}\left[\left[\mathfrak{a},\mathfrak{b}\right],\mathfrak{a}\right] + \cdots}_{\text{Error terms}}\Big), \tag{5}$$

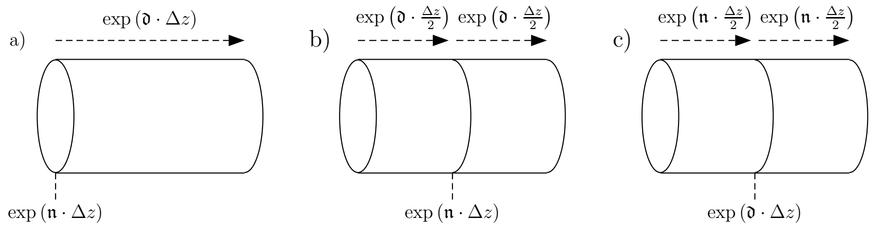
a)
$\exp(\mathfrak{d}\cdot\Delta z)$
$\exp(\mathfrak{n}\cdot\Delta z)$
b)
$\exp(\mathfrak{d}\cdot\frac{\Delta z}{2})$ $\exp(\mathfrak{d}\cdot\frac{\Delta z}{2})$
$\exp(\mathfrak{n}\cdot\Delta z)$
c)
$\exp(\mathfrak{n}\cdot\frac{\Delta z}{2})$ $\exp(\mathfrak{n}\cdot\frac{\Delta z}{2})$
$\exp(\mathfrak{d}\cdot\Delta z)$

Figure 1: Illustration of one step of the Split-Step Fourier Method in the standard a) and the symmetric variant with the linear operator b) and the nonlinear operator c) evaluated in two steps.

with $[\mathfrak{a},\mathfrak{b}] = \mathfrak{a}\mathfrak{b} - \mathfrak{b}\mathfrak{a}$. If one applies (4) instead of (3) then $\mathfrak{a} = \mathfrak{d}(t)\Delta z$ and $\mathfrak{b} = \mathfrak{n}(z,t)\Delta z$. The dominant contribution to the error is given by the first term $\frac{1}{2}[\mathfrak{a},\mathfrak{b}] = \frac{1}{2}\Delta z^2 [\mathfrak{d},\mathfrak{n}]$ and hence is $O(\Delta z^2)$, or second order in Δz. By saying that an error is $O(\Delta z^k)$ we mean that is is bounded by $C\Delta z^k$ given a certain constant C.

In order to reduce the error of the split-step scheme it is useful to apply a symmetric approach, in which either the linear or nonlinear operator is split into two parts, which are applied before and after the respective other operator. This is illustrated in Fig. 1b) and c). To show the outcome of the symmetric scheme we start from (5) and summarize all errors beyond second order as $O(\Delta z^3)$,

$$\exp\left(\mathfrak{n}\frac{\Delta z}{2}\right) \cdot \exp(\mathfrak{d}\Delta z) = \exp\left(\mathfrak{n}\frac{\Delta z}{2} + \mathfrak{d}\Delta z + \frac{\Delta z^2}{4}[\mathfrak{n},\mathfrak{d}] + O(\Delta z^3)\right) = \exp(\mathfrak{c}). \tag{6}$$

Using this result one can obtain

$$\begin{aligned}\exp(\mathfrak{c}) \cdot \exp\left(\mathfrak{n}\frac{\Delta z}{2}\right) &= \exp\left(\mathfrak{n}\frac{\Delta z}{2}\right) \cdot \exp(\mathfrak{d}\Delta z) \cdot \exp\left(\mathfrak{n}\frac{\Delta z}{2}\right)\\ &= \exp\left(\mathfrak{c} + \mathfrak{n}\frac{\Delta z}{2} + \frac{\Delta z}{4}[\mathfrak{c},\mathfrak{n}] + O(\Delta z^3)\right)\\ &= \exp\left(\mathfrak{n}\Delta z + \mathfrak{d}\Delta z + O(\Delta z^3)\right).\end{aligned} \tag{7}$$

Please note that the time and position dependence of the operators $\mathfrak{n}(z,t)$ and $\mathfrak{d}(t)$ is omitted for the sake of brevity. Most publications and text books, e.g. [Agr07], describe the symmetric split-step according to Fig. 1b). By exchanging the operators in the above derivation one can verify that this also leads to an error of third order in Δz. However, the evaluation of the linear operator is usually more time consuming and therefore it is beneficial to only evaluate it once per symmetric split-step rather than twice.

The linear operator can be implemented in the Fourier-Domain by using the correspondence $\partial/\partial t \circ\!\!-\!\!\bullet\, j\omega$. Hence this scheme is commonly referred to as *Split-Step Fourier Method*. The application of the linear operator to the signal $u(z,t)$ can be written as

$$\exp(\mathfrak{d}\Delta z)\, u(z,t) = \mathcal{F}^{-1}\left\{(\exp(\mathfrak{D}\Delta z)\,\mathcal{F}\{u(z,t)\}\right\}, \tag{8}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier-transform and its inverse, respectively, and $\mathfrak{D}$ is the linear operator in the frequency domain. The computational-effort is dominated by the Fourier-transforms.

The nonlinear operator can be realized in the time-domain by applying a phase rotation (2) to the signal. The term $\mathfrak{n}(u(z,t)) \cdot \Delta z$ is a coarse approximation of the accumulated nonlinearity over the segment of length Δz. A more accurate result will be achieved at the same step-width Δz if the evolution of the signal along Δz is used to determine the nonlinear operator. This can for example achieved by using the Trapezoid-rule in combination with iterative refinement [Agr07]. However, the number of additional operations increases simulation time. Hence, it can be beneficial to omit the iterative approach and to rather decrease the step-size to preserve the accuracy of the scheme.
So far we have only considered one step. In order to compute the evolution of the signal along the fiber the symmetric split-step is repeatedly carried out

until the end of the fiber is reached. Using a constant Δz is inefficient. There are various approaches to choosing the appropriate distribution of step-sizes Δz and a good overview can be found in [SHZM03]. One approach, which directly reflects the dependence of $\mathfrak{n}$ on the instantaneous power of the signal is the *Nonlinear Phase-Rotation Method*. In this method an estimate of the significance of the nonlinear operator is found based on the nonlinear phase shift, which is produced by an application of the nonlinear operator. In order to keep the strength of the nonlinearity in each segment below a certain level one can upper bound the nonlinear phase-shift by choosing

$$\Delta z = \frac{\phi_{\mathrm{NL,max}}}{\gamma \max_{\forall t}\left\{\left|u\left(z,t\right)\right|^2\right\}} = \min_{\forall t}\left\{\frac{\phi_{\mathrm{NL,max}}}{\gamma\left|u\left(z,t\right)\right|^2}\right\}. \tag{9}$$

The control-parameter $\phi_{\mathrm{NL,max}}$ for this method has to be adjusted carefully. If it is chosen very small then this may lead to large number of steps. However, if the selected value of $\phi_{\mathrm{NL,max}}$ is too large then the result will become unreliable. The algorithm, which implements the symmetric SSFM is shown in Fig. 2.

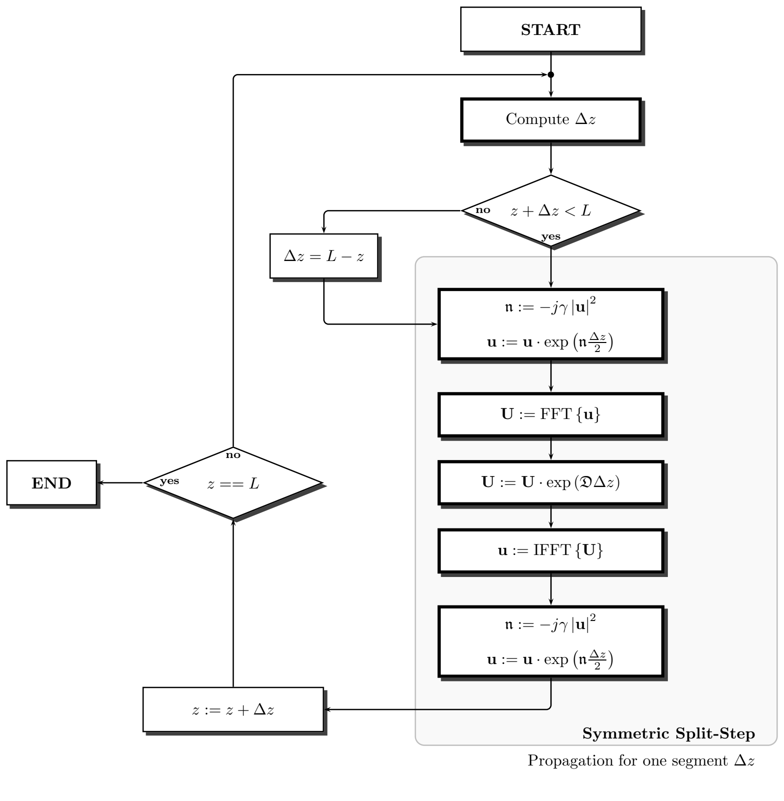
START
Compute Δz
no
$z + \Delta z < L$
yes
$\Delta z = L - z$
$\mathfrak{n} := -j\gamma\left|\mathbf{u}\right|^2$
$\mathbf{u} := \mathbf{u}\cdot\exp\left(\mathfrak{n}\frac{\Delta z}{2}\right)$
$\mathbf{U} := \mathrm{FFT}\left\{\mathbf{u}\right\}$
$\mathbf{U} := \mathbf{U}\cdot\exp\left(\mathfrak{D}\Delta z\right)$
$\mathbf{u} := \mathrm{IFFT}\left\{\mathbf{U}\right\}$
$\mathfrak{n} := -j\gamma\left|\mathbf{u}\right|^2$
$\mathbf{u} := \mathbf{u}\cdot\exp\left(\mathfrak{n}\frac{\Delta z}{2}\right)$
END
yes
no
$z == L$
$z := z + \Delta z$
Symmetric Split-Step
Propagation for one segment Δz

Figure 2: Flow diagram of the symmetric Split-Step Fourier Method for solving the scalar Nonlinear-Schrödinger Equation according to Fig.1c). All arithmetic operations are on each element of the signal vector.

3 Parallel Implementation of the Split-Step Fourier Method using GPU and CUDA

Over the past few years the increase of the raw computational power delivered by GPUs has exceeded that of central processing units (CPU) by far [Gee05]. This development was mainly driven by the video gaming industry, which demanded more and more performance to facilitate realistic graphic effects and high frame rates. However, it is also the application of graphics rendering itself, which has led to a highly parallel chip-architecture to provide this performance at comparably low clock rates. It is interesting to note that GPUs hence naturally have been designed in a parallel architecture, whereas CPUs were traditionally designed in a serial architecture.

Motivated by the computational power that GPUs can supply, researchers and software developers have tried to use these devices for purposes other than graphics rendering, a trend called general purpose GPU processing (GPGPU). One aspect that has made GPGPU difficult was that the specific hardware had to be accessed using a graphics programming interface, e.g. OpenGL. This problem has now been addressed by NVIDIA, who have introduced a general purpose programming interface called compute unified device architecture (CUDA) for their GPUs [NVI09][Hal08] in 2006. Programs for CUDA can be written using C as a high-level programming language.

The structure of an NVIDIA GPU suitable for GPGPU is shown on the left side of Fig. 3. The fundamental components are the so called streaming multiprocessors (SM). Each of these streaming multiprocessors contains M processors/cores, which are capable of floating point operations. In addition to that, the SMs contain different types of memory, the most important being the shared memory, which all of the cores in a SM can access in parallel and use to exchange information in an extremely fast way. As an example, the GT200b chip features 30 SMs each containing 8 single precision processors and one double precision processor. This amounts to 240 single precision and 30 double precision cores in total. The aim is to offload parallel computations from the

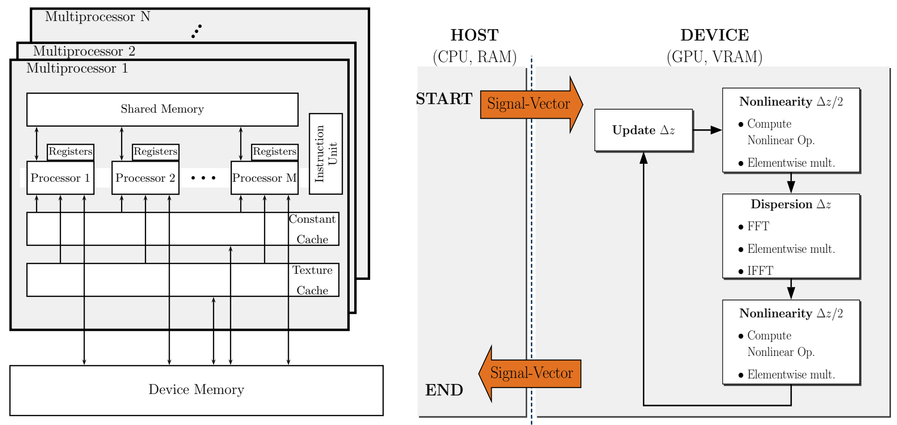

Figure 3: The architecture of a CUDA enabled graphics processing unit is shown on the left. The right diagram schematically depicts how the computational steps are carried out on the CPU and the GPU.

CPU onto the GPU to exploit this massively parallel processor and hence obtain reduced computation time.
In the previous section the SSFM was revisited. A significant amount of the operations performed in the SSFM lends itself to a parallel implementation. This has already been highlighted in [ZRZB99]. In particular, the parallel implementation of the linear operator in the Fourier domain as well as the nonlinear parameter in the time domain is straightforward. The signal vector $\mathbf{u}$ can be split up into N_{proc} blocks containing N/N_{proc} elements. The element-wise multiplications can then be performed in parallel by N_{proc} processing units.

The most time consuming task in the SSFM is the Fourier-transform. The parallel computation of the Discrete Fourier Transform (DFT) is not as trivial as the parallelization of the other operators since each element in the output-vector of the DFT depends on all elements of the input vector. Therefore the computation has to be divided into multiple stages. The different variants of the Fast Fourier Transform (FFT) algorithm are all based on a divide-and-conquer approach [CT65]. The Radix-2 algorithm for example separates the computation of an N point FFT calculation into $\log_2 N$ steps. Optimized FFT routines are available for NVIDIA GPUs as a part of CUDA [NVI08].

The schematic showing the implementation of the SSFM using a GPU is shown on the right hand side of Fig. 3. In the following, the CPU/RAM will be referred to as *Host* and the GPU/VRAM as *Device*. At the beginning of the program, the signal vector is stored on the *Host*. In order to offload the computation to the GPU in a first step memory has to be allocated and the signal vector is then transferred onto the *Device*. Transfers between *Host* and *Device* are relatively slow and therefore should be avoided whenever possible. Since we are trying to minimize the number of FFT operations we are using the scheme shown in Fig. 1c).

Now the step size for the first symmetric step has to be found. After that the first nonlinear step over $\Delta z/2$ is computed, which involves computation of the nonlinear operator and a subsequent element wise multiplication with the signal vector. Then the signal is transformed into the frequency domain for the dispersive step by using [NVI08]. After another element-wise multiplication the signal vector is transformed back into the time-domain, where the second nonlinear step over $\Delta z/2$ concludes the symmetric step. This procedure is iterated as shown in Fig. 2. When the end of the fiber segment is reached the signal is copied back onto the *Host*.

4 Evaluation of Performance and Accuracy

In order to determine the achievable savings in computation time we have carried out simulations of an optical transmission system for a program using only the CPU and for a program, which offloads the computation of the SSFM to the GPU.

The hardware used for the simulation was a Fujitsu Siemens Workstation M460 equipped with a Core 2 Quad processor Q9650 (3 GHz, 6MB cache, 4 cores) and 8 GB RAM. The graphics hardware used for the simulation was manufactured by Zotac and features a GT200b chip by NVIDIA, 1024 GB GDDR3 RAM. The chip contains a total of 240 cores clocked at 648 MHz. The additional cost for the graphics hardware was a fraction of the cost for the initial hardware. The operating system used on the computer was OpenSUSE Linux 11.1 64bit equipped with CUDA 2.3 and the corresponding GPU driver of version 190.18. The SSFM code was implemented in the C programming language in both cases. However, the GPU supported version also used the CUDA extensions. The FFT operation on the CPU was implemented by the FFTW library [FJ08] (v3.2.1) and on the GPU the optimized CUFFT library was used. The other components of the simulation were implemented in MATLAB.[1]. Both implementations of the SSFM were called from MATLAB via the mex-interface. The system used for comparison of the two approaches was a 10.7 Gb/s RZ-DPSK system as depicted in Fig. 4. Pulses corresponding to 50% return-to-zero (RZ) are carved onto the output of a CW-laser and subsequently the pulse train is modulated according to a de Bruijn pseudorandom binary sequence (DBBS). The corresponding optical signal is amplified to P_{in} and then launched into the link, which comprises 80km spans of standard single-mode fiber (D=16ps/nm/km, γ=1.3W^{-1}km^{-1}, α=0.2dB/km), the dispersion of which is fully compensated by using dispersion compensating fiber (DCF) (D=-80ps/nm/km, γ=4.8W^{-1}km^{-1}, α=0.5dB/km, L=16km). After each span an ideal amplifier fully compensates the loss the signal has experienced. The temporal resolution is 32 points per symbol.

4.1 Performance of the Program based on the Graphics Processor

The first series of simulations was performed to evaluate the potential time saving to be achieved by using the GPU. The simulation of transmission over N_{span}=30 spans was carried out for increasing order of the DBBS both for the program using only the CPU and for the GPU assisted code. We have evaluated two different versions of the SSFM on the GPU: one version uses single precision computations and the other one uses double precision. The single precision version was investigated since the larger number of single precision cores on the GPU promises faster computations in comparison to double precision. After the simulation the time consumed by the SSFM program was measured and compared for the two cases. The results are shown in Fig. 5 in terms of the relative speed-up obtained from using the GPU. The first observation to be made is that a significant speed-up can be achieved. Longer signal vectors

[1]It will also be beneficial to implement other functions such that they utilize the power of the GPU. However, in this work we focus on the fiber simulation only.

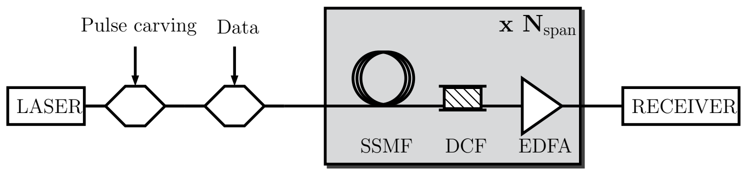

Figure 4: Test system for the simulation of 10.7 Gb/s RZ-DPSK over N_{span} fully dispersion compensated spans for $\phi_{\text{max}} = 10^{-3}$

clearly result in a larger advantage for the GPU assisted program. This is due to the fact that more elements lead to a better utilization of the GPU. For sequences shorter than order 7 there can be configurations, in which the CPU based implementation works faster. However, very short test sequences are not sufficient in modern optical communication systems. Hence, the case in which the GPU implementation gives inferior performance practically does not exist. The results also show that higher input power leads to increased acceleration. This can be explained by the fact that higher P_{in} on average implies a larger number of steps. Therefore the number of the operations carried out on the graphics card in parallel more and more outweighs the computational overhead incurred on every span, e.g. calling the mex-function and up-/downloading the signal onto/from the graphics card's memory. The dramatic increase of the GPU assisted acceleration for larger than order 13 sequences can be explained by the reduced performance of the CPU program as the size of the signal vector approaches the size of the CPU cache.

Due to the larger number of single precision cores the single precision GPU assisted program provides much larger speed up factors. The highest recorded speed-up factors were 80 for single precision and 25 for double precision for an order 15 sequence and a span input power of $P_{\mathrm{in}} = 6\mathrm{dBm}$. This means that a set of simulations, which normally would have taken almost a month using the CPU only can be finished in only one day by using the GPU in double precision and less than eight hours for single precision.

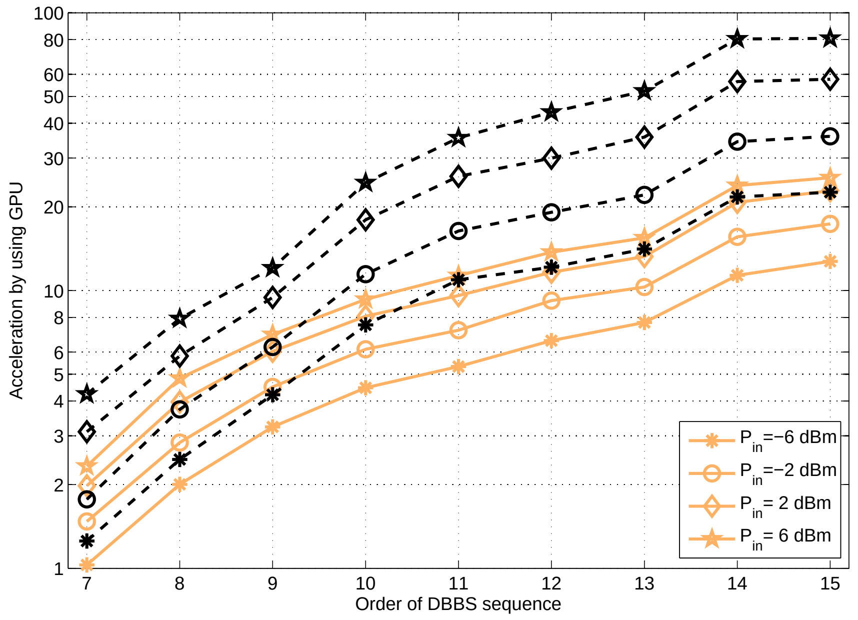

Figure 5: Acceleration of simulation versus order of the test sequence for transmission of 10.7 Gb/s RZ-DPSK over $N_{\mathrm{span}} = 30$ spans and $\phi_{\max} = 5 \cdot 10^{-3}$. Dashed lines correspond to single-precision and solid lines indicate the double-precision GPU assisted program.

4.2 Accuracy of the Program based on the Graphics Processor

The objective of the second series of simulations was to determine the computational accuracy of the GPU based approach, in particular of the single precision SSFM. In the same way as [SHZM03] we have used the normalized *root mean squared error* (RMSE) as a measure of accuracy.
The RMSE ϵ can be expressed as

$$\epsilon = \frac{\|u(L,t) - \hat{u}(L,t)\|}{\|\hat{u}(L,t)\|}, \tag{10}$$

where $u(L,t)$ is the result of the simulation and $\hat{u}(L,t)$ is the true result. Due to lack of an analytical $\hat{u}(L,t)$ we have used the result of the purely CPU based simulation with $\phi_{\max} = 10^{-6}$ as the true result. Once again, the simulation of transmission over 30 spans was carried out both using the GPU assisted code and the program, which uses only the CPU. The length of the DBBS was chosen to be 64 and $\phi_{\max} \in [0.0008, \ldots, 0.05]$ was varied to tune the accuracy of the SSFM. The result of the simulations is shown in Fig. 6 in terms of ϵ for a given number of FFT operations N_{FFT} and input power levels $P_{\text{in}} \in [-6, 0, 6]$dBm.

The first observation that can be made from these results is that for the program using only the CPU the accuracy improves continuously with increasing N_{FFT}, i.e. decreasing $\phi_{\max}$ and hence decreasing step-size Δz [SHZM03]. It is also evident that the number of FFT operations to obtain a certain ϵ grows for increasing power levels. The accuracy for the double precision GPU assisted program completely overlaps with CPU only program for the range of investigated $\phi_{\max}$. The results for the single precision GPU assisted program, however, do not continuously improve with increasing N_{FFT}. For larger values of $\phi_{\max}$ the results completely overlap with the results of the CPU only simulation until they reach a minimum beyond which ϵ starts to increase again.

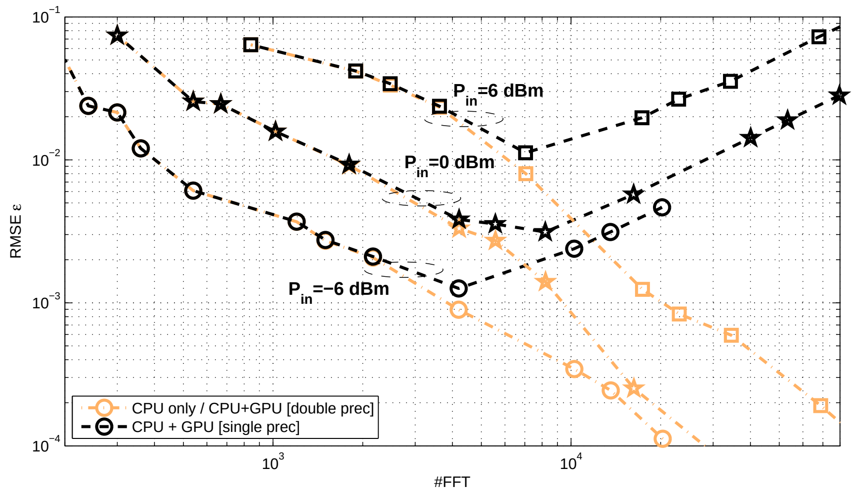

Figure 6: Simulation accuracy for the 10.7 Gb/s RZ-DPSK transmission system and a link comprising 30 spans. RMSE is the measure of accuracy. The double precision GPU results overlap with the results obtained by using only the CPU.

An explanation for the dependence of the RMSE on N_{FFT} can be given by modeling the deviation from the true result as a noise process, i.e.

$$u\,(L,t)=\hat{u}\,(L,t)+n_{\mathrm{NL}}(L,t)+n_{\mathrm{C}}(L,t), \tag{11}$$

where $n_{\mathrm{NL}}(L,t)$ and $n_{\mathrm{C}}(L,t)$ are noise terms due to the separation of linear and nonlinear operators and the numerical inaccuracy, respectively. The RMSE ϵ is then given by

$$\epsilon=\frac{\|u\,(L,t)-\hat{u}\,(L,t)\|}{\|\hat{u}\,(L,t)\|}=\frac{\|n_{\mathrm{NL}}(L,t)+n_{\mathrm{C}}(L,t)\|}{\sqrt{P_{\mathrm{in}}}}=\frac{\sigma_{\mathrm{NL}}+\sigma_{\mathrm{C}}}{\sqrt{P_{\mathrm{in}}}}, \tag{12}$$

where σ_{NL} and σ_{C} are the standard deviations of the equivalent noise distributions. The RMSE is formed by the two contributions due to the inaccuracy for small step-size σ_{NL} and due to the computational error σ_{C}. The local error of the symmetric split-step is $O(\Delta z^3)$, which means that it is smaller or equal to $C\Delta z^3$. For the same step size Δz the error is larger for stronger nonlinearity, which is caused by larger power levels. The cubic dependence on the signal is reflected by the factor $P_{\mathrm{in}}\sqrt{P_{\mathrm{in}}}$. The accumulated error for applying the SSFM for an entire link is upper bounded by

$$n_{\mathrm{NL,max}}(L,t)=P_{\mathrm{in}}\sqrt{P_{\mathrm{in}}}\sum_{n=0}^{N-1}C_n\Delta z_n^3\approx P_{\mathrm{in}}\sqrt{P_{\mathrm{in}}}C'N_{\mathrm{FFT}}\overline{\Delta z}^3, \tag{13}$$

where $\overline{\Delta z}=\frac{1}{N}\sum_{n=0}^{N-1}\Delta z_n$ is the average step-size and C' is a constant. If we assume that the number of FFT operations is given by $N_{\mathrm{FFT}}=2\cdot L/\overline{\Delta z}$ then the length of one average step can be expressed by $\overline{\Delta z}=2\cdot L/N_{\mathrm{FFT}}$, which means that the mean contribution to the RMSE at the end of the link is

$$\sigma_{\mathrm{NL}}\approx C'\cdot\frac{P_{\mathrm{in}}\sqrt{P_{\mathrm{in}}}L^3}{N_{\mathrm{FFT}}^2} \tag{14}$$

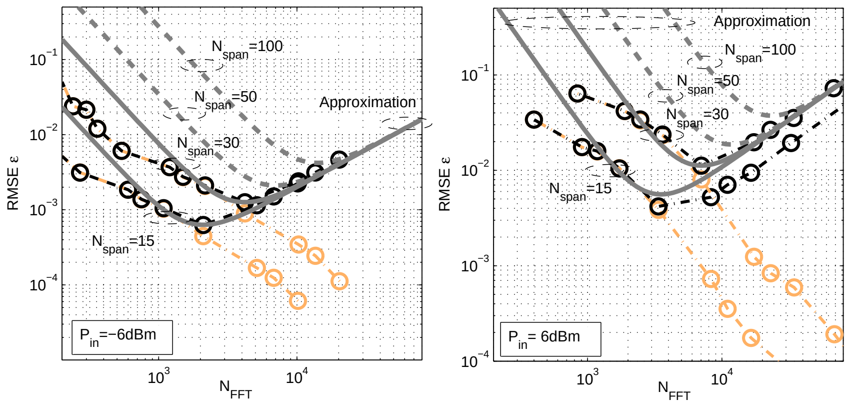

Figure 7: Simulation accuracy for the 10.7 Gb/s RZ-DPSK transmission system and links comprising 15 and 30 spans. RMSE is the measure of accuracy. The analytical approximation of the RMSE evolution is given by the thick solid lines with predictions for $N_{\mathrm{span}}\in\{50,100\}$ in thick dashed lines. The double precision GPU results overlap with the results obtained by CPU only.

and therefore $\propto N_{\mathrm{FFT}}^{-2}$ and $\propto P_{\mathrm{in}}\sqrt{P_{\mathrm{in}}}$ where a factor 8 has been absorbed by the redefinition
$C' := 8C'$. The dominant contribution to the computational error is caused by the single precision FFT operations. The average error due to one single precision FFT operation is ϵ_{FFT}. It is constant for a signal vector with unit power and a given size. Hence, the global contribution to the RMSE is proportional to $\sqrt{P_{\mathrm{in}}}$ and can be expressed as

$$\sigma_{\mathrm{C}} \approx C_2' \cdot \sqrt{P_{\mathrm{in}}} \cdot \epsilon_{\mathrm{FFT}} \cdot N_{\mathrm{FFT}}. \tag{15}$$

Using these results the RMSE can be approximated as

$$\epsilon_{\max} \approx C' \cdot \frac{P_{\mathrm{in}} L^3}{N_{\mathrm{FFT}}^2} + C_2' \cdot \epsilon_{\mathrm{FFT}} \cdot N_{\mathrm{FFT}}, \tag{16}$$

which implies a v-shape in the log-log Domain with a minimum at $N_{\mathrm{FFT,min}} = \sqrt[3]{\frac{2C' P_{\mathrm{in}} L^3}{C_2' \epsilon_{\mathrm{FFT}}}}$.

Based on the simulation results for one configuration C' and C_2' can be determined. This was done for the case of $N_{\mathrm{span}} = 30$ and $P_{\mathrm{in}} \in \{-6, 6\}$dBm. The resulting analytical approximation of the RMSE is shown in Fig. 7.

For a smaller number of FFT operations, which corresponds to longer step sizes, the gradient of the RMSE is not as large as predicted by the theory. This can be explained by the inaccuracy of the nonlinear operator. For smaller step-size the results approach the predicted third order dependence on Δz more closely. Another interesting aspect to note is that the optimal step-size control parameter $\phi_{\max}$ does change with P_{in} but does remain constant for varying number of spans. The analytical approximation can also be used to predict the error to be encountered for links comprising more than 30 spans. The corresponding result of the RMSE for 50 and 100 spans is also shown in Fig. 7.

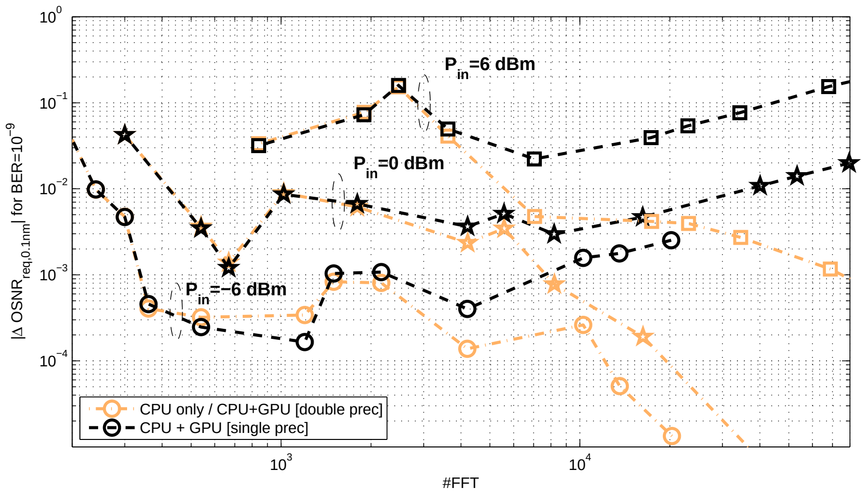

Figure 8: Accuracy for the simulation of the 10.7 Gb/s RZ-DPSK system over 30 spans measured in terms of the deviation from the required $\mathrm{OSNR}_{0.1\mathrm{nm}}$ for a Bit-Error-Rate of 10^{-9} in dB obtained for $\phi_{\max} = 10^{-6}$.

One use of the approximation is to predict the maximum number of spans, which can be simulated before a certain RMSE is exceeded.

The RMSE is a widely accepted measure for accuracy. From a system designer's point of view, however, it is more useful to see the difference of the results in terms of a system figure of merit, e.g. the required optical signal-to-noise ratio (OSNR) for a reference bandwidth of 0.1nm to obtain a certain bit-error-rate (BER). We have used the output of the fiber simulation for both the CPU and the GPU assisted SSFM program to determine the required OSNR for a BER of 10^{-9} in dB. The receiver model consisted of a second order optical Gaussian filter with 3dB bandwidth $B_{\mathrm{opt}} = 3{\cdot}R_b$ followed by an ideal photo diode and a fifth order electrical Bessel filter with 3dB bandwidth $B_{\mathrm{ele}} = 0.7 \cdot R_b$. The bit-error-probability was calculated using a semi-analytical approach [For00] for different values of OSNR and from these results the required OSNR was inferred. Subsequently, we have compared the true required OSNR based on the calculation using the CPU only with $\phi_{\max} = 10^{-6}$ to the values obtained through computations using the CPU and the GPU, respectively. The difference $\Delta\mathrm{OSNR}_{\mathrm{req}}$ in dB for $N_{\mathrm{span}} = 30$ spans is depicted in Fig. 8.

One first thing that can be noticed is the correlation between the evolution of the RMSE in Fig. 6 and the evolution of the difference $|\Delta\mathrm{OSNR}_{\mathrm{req}}|$in Fig. 8. The resulting $|\Delta\mathrm{OSNR}_{\mathrm{req}}|$ in dB is relatively small. The fluctuation for the differences smaller than 0.01dB can be explained by the limited accuracy of the estimation of the required OSNR. Practically these very small differences are insignificant. There are only two intervals of N_{FFT}, in which $P_{\mathrm{in}} = 6$dBm, for which $|\Delta\mathrm{OSNR}_{\mathrm{req}}|$ exceeds 0.1dB.

In the left interval this is caused by the inaccuracy due to an insufficient step-size. Therefore in this interval the standard program and the GPU assisted version return the same results. In the right interval the inaccuracy stems from the accumulated error due to single precision computations. However, even for $N_{\mathrm{FFT}} = 8{\cdot}10^4$ when RMSE $\epsilon \approx 0.1$ the error is only 0.2dB.

One can use the analytical approximation for the RMSE to predict when 0.1dB difference are exceeded. By looking at Fig. 7 it can be estimated that for $P_{\mathrm{in}} = 6$dBm the simulation of more than $N_{\mathrm{span}} = 100$ using the single precision version of the SSFM is possible.

5 Conclusion

The simulation of fiber-optic communication links can be a time consuming task. In this paper we have shown that the Split-Step Fourier Method, which is commonly used to calculate the signal evolution in single-mode fiber, can be significantly accelerated by the use of commodity graphics hardware. The implementation of the SSFM on a GT200b GPU by NVIDIA was compared to an implementation, which relied only upon a state-of-the-art CPU. The additional cost for the graphics hardware was a fraction of the cost of the original workstation. The simulation of an optical communication system was carried out to measure the performance.

We have found substantial acceleration compared to the CPU based implementation in all cases we have investigated both for the single precision and for the double precision implementation on the GPU. The peak acceleration was found to be ≈ 80 for single precision and ≈ 25 for double precision in our test scenario.

In addition to the speed measurements we have also conducted simulations to evaluate the accuracy of the GPU based approach. The simulations reveal that there is practically no difference between the results of the double precision GPU computation and the computations based on the CPU only. If the computations on the GPU are computed in single precision then the accumulated error, which mostly depends on the number of FFT/IFFT operations, can produce inaccurate results. Nevertheless, single precision computation gives sufficiently accurate results in many scenarios. In addition to that, mixed accuracy computations could be combined in a suitable manner for example by using the stratified Monte Carlo sampling technique recently proposed in [SRBB09].

We have concentrated on the scalar version of the SSFM in this work. We have also implemented the coarse-step method [EJMGB92] to include polarization effects. The advantages of the GPU based implementation are preserved and even further emphasized in this case.

References

[Agr07] G. P. Agrawal. *Nonlinear Fiber Optics*. Academic Press, 4 edition, 2007.

[CGH09] Leonardo D. Coelho, Oscar Gaete, and Norbert Hanik. An algorithm for global optimization of optical communication systems. *AEU - International Journal of Electronics and Communications*, 63(7):541 – 550, 2009.

[CMG$^+$09] Leonardo Didier Coelho, L. Molle, D. Gross, Norbert Hanik, Ronald Freund, C. Caspar, E.-D. Schmidt, and B. Spinnler. Modeling Nonlinear Phase Noise in Differentially Phase-Modulated Optical Communication Systems. *Optics Express*, 17(5):32263241, March 2009.

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computations*, 19(90):297–301, April 1965.

[EJMGB92] Stephan G. Evangelides Jr., Linn F. Mollenauer, James P. Gordon, and Neal S. Bergano. Polarization Multiplexing with Solitons. *IEEE Journal of Lightwave Technology*, 10(1):28–35, January 1992.

[FJ08] M. Frigo and Steven G. Johnson. Fastest fourier transform in the west (fftw). Webpage, 2008.

[For00] Enrico Forestieri. Evaluating the Error Probability in Lightwave Systems with Chromatic Dispersion, Arbitrary Pulse Shape and Pre-and Postdetection Filtering. *IEEE Journal of Lightwave Technology*, 18(11):1493–1503, November 2000.

[Gee05] David Geer. Taking the graphics processor beyond graphics. *Journal of the IEEE Computer Society*, 9(9):14–16, September 2005.

[Hal08] Tom R. Halfhill. Parallel Processing with CUDA. *Microprocessor Report*, pages 1–8, January 2008.

[Mag54] Wilhelm Magnus. On the Exponential Solution of Differential Equations for a Linear Operator. *Communications on Pure and Applied Mathematics*, 7:649–673, 1954.

[NVI08] NVIDIA Corp. *CUDA - CUFFT Library, Version 2.1*, April 2008.

[NVI09] NVIDIA Corp. *NVIDIA CUDA - Programming Guide*, 2009.

[SHZM03] O. V. Sinkin, R. Holzlöhner, J. Zweck, and C.R. Menyuk. Optimization of the Split-Step Fourier Method in Modeling Optical-Fiber Communications Systems. *IEEE Journal of Lightwave Technology*, (1):61–68, January 2003.

[SRBB09] P. Serena, N. Rossi, M. Bertolini, and A. Bononi. Stratified Sampling Monte Carlo Algorithm for Efficient BER Estimation in Long-Haul Optical Transmission Systems. *IEEE Journal of Lightwave Technology*, 27(13):2404–2410, July 2009.

[WEG$^+$04] L.K. Wickham, R.-J. Essiambre, A.H. Gnauck, P.J. Winzer, and A.R. Chraplyvy. Bit pattern length dependence of intrachannel nonlinearities in pseudolinear transmission. *Photonics Technology Letters, IEEE*, 16(6):1591–1593, June 2004.

[WM62] G. H. Weiss and A. A. Maradudin. The Baker-Hausdorff Formula and a Problem in Crystal Physics. *Journal of Mathematical Physics*, 3:771–777, 1962.

[ZRZB99] S. Zoldi, V. Ruban, A. Zenchuk, and S. Burtsev. Parallel Implementation of the Split-step Fourier Method for Solving Nonlinear Schrödinger Systems. *SIAM News*, pages 1–5, 1999.