The nonequispaced FFT on graphics processing units

Susanne Kunis* and **Stefan Kunis****

University of Osnabrueck, 49076 Osnabrueck

Without doubt, the fast Fourier transform (FFT) belongs to the algorithms with large impact on science and engineering. By appropriate approximations, this scheme has been generalized for arbitrary spatial sampling points. This so called nonequispaced FFT is the core of the sequential NFFT3 library and we discuss its computational costs in detail. On the other hand, programmable graphics processing units have evolved into highly parallel, multithreaded, manycore processors with enormous computational capacity and very high memory bandwidth. By means of the so called Compute Unified Device Architecture (CUDA), we parallelized the nonequispaced FFT using the CUDA FFT library and a dedicated parallelization of the approximation scheme.

1 Introduction

Fast Fourier transforms (FFTs) have been generalized to arbitrary sampling situations, see [1–4] for its mathematical foundation and [5–9] for recent surveys. A convenient approach is to define the discrete Fourier transform as the evaluation of a trigonometric polynomial, given by its Fourier coefficients, at equispaced spatial nodes. This can then be generalized to arbitrary nodes using a dedicated approximation scheme as discussed in Section 2. Following prior approaches in [10, 11], we accelerate the nonequispaced FFT by using commodity graphics hardware in Section 3. The last section of this short note is dedicated to a performance comparison of our implementation in CUDA against the NFFT3 library [12]. Following the widely accepted concept of reproducible research, our implementation is freely available at [13].

2 Nonequispaced Fast Fourier Transform

For ease of notation, we consider the onedimensional case $d = 1$ first. Following standard conventions [14, 15] the (forward) discrete Fourier transform (DFT) is defined as the calculation of the sums

$$f_j = \sum_{k=0}^{N-1} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} jk/N}, \qquad j = 0, \dots, N-1,$$

for $N \in \mathbb{N}$ and given coefficients $\hat{f}_k \in \mathbb{C}$. The computation of this transform requires $\mathcal{O}(N \log N)$ arithmetic operations by means of the fast Fourier transform (FFT). We define the nonequispaced fast Fourier transform (NFFT) as the fast and approximate evaluation of the trigonometric polynomial

$$f(x) = \sum_{k \in I_N} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} kx}, \qquad j = 0, \dots, M-1, \tag{1}$$

at nonequispaced nodes $x_j \in \mathbb{T} = [-\frac{1}{2}, \frac{1}{2}]$, where $I_N = \{-\frac{N}{2}, \dots, \frac{N}{2} - 1\}$ denotes the set of frequencies.

The idea of the NFFT is to use a standard FFT in combination with an approximation scheme that is based on a window function $\varphi : \mathbb{R} \to \mathbb{R}$ which is mutually well localized in the spatial and frequency domains. The first step is to approximate the trigonometric polynomial f of degree N in (1) by a linear combination

$$s_1(x) = \sum_{l \in I_n} g_l \, \tilde{\varphi}\left(x - \frac{l}{n}\right), \tag{2}$$

of shifted one-periodic window functions $\tilde{\varphi}(x) = \sum_{r \in \mathbb{Z}} \varphi(x + r)$, where $n = \sigma N$ for some oversampling factor $\sigma > 1$ determines the length of the ordinary FFT used below. Switching to the frequency domain yields

$$s_1(x) = \sum_{k \in I_n} \hat{g}_k \, \hat{\varphi}(k) \, \mathrm{e}^{-2\pi \mathrm{i} kx} + \sum_{r \in \mathbb{Z} \setminus \{0\}} \sum_{k \in I_n} \hat{g}_k \, \hat{\varphi}(k + nr) \, \mathrm{e}^{-2\pi \mathrm{i}(k+nr)x}$$

* Corresponding author E-mail: sukunis@uos.de, Phone: +00 49 541 969 2471
** Corresponding author E-mail: skunis@uos.de, Phone: +00 49 541 969 2538

with Fourier coefficients

$$\hat{g}_k = \sum_{l \in I_n} g_l \, \mathrm{e}^{2\pi \mathrm{i} \frac{kl}{n}}, \qquad \hat{\varphi}(k) = \int_{\mathbb{R}} \varphi(x) \, \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x \tag{3}$$

A comparison of (1) and (2) suggests to define

$$\hat{g}_k = \begin{cases} \frac{\hat{f}_k}{\hat{\varphi}(k)} & \text{for } k \in I_N, \\ 0 & \text{for } k \in I_n \backslash I_N, \end{cases} \tag{4}$$

assuming that $\hat{\varphi}(k)$ is small for $|k| \geq n - \frac{N}{2}$. This first approximation causes an aliasing error. The values g_l can now be computed by an FFT of length n, applying the Fourier inversion theorem to (3). Assuming now a strong localization in the spatial domain, we truncate the sum (2) and replace the one-periodic window functions by the original one to have the final approximation

$$s(x_j) = \sum_{l \in I_{n,m}(x_j)} g_l \, \varphi\left(x_j - \frac{l}{n}\right) \tag{5}$$

where $I_{n,m}(x_j) = \{l \in I_n : nx_j - m \leq l \leq nx_j + m\}$ has cardinality at most $2m+1$ and collects indices of the neighboring grid points of x_j.Here, we consider the Gaussian window function

$$\varphi(x) = (\pi b)^{-1/2} \, \mathrm{e}^{-\frac{(nx)^2}{b}}, \qquad b = \frac{2\sigma}{2\sigma - 1} \frac{m}{\pi},$$

which leads to the error estimate

$$|f(\boldsymbol{x}_j) - s(\boldsymbol{x}_j)| \leq 4 \, \mathrm{e}^{-m\pi(1-1/(2\sigma-1))} \sum_{k \in I_n} |\hat{f}_k|$$

and thus the choice $m = C|\log \varepsilon|$ assures accuracy ε of the NFFT. Moreover, this window function allows for the factorization

$$\varphi\left(x_j - \frac{l'}{n}\right) = (\pi b)^{-1/2} \, \mathrm{e}^{-\frac{(nx_j - u)^2}{b}} \left(\mathrm{e}^{-\frac{2(nx_j - u)}{b}}\right)^l \mathrm{e}^{-\frac{l^2}{b}},$$

where $u = \min I_{n,m}(x_j)$ and $l = 0, \ldots, 2m$. The first factor and the exponential within the brackets are a constant for each fixed node x_j. Once we have evaluated the second exponential, its lth power can be computed by repeated multiplications only. Furthermore, the last exponential is independent of x_j such that these $2m+1$ different values need to be precomputed only once – usually a negligible amount. Thus, it is sufficient to store or evaluate $2M$ exponentials and this cute trick has been termed fast Gaussian gridding [8].

The described approximation method can be generalized to arbitrary dimensions in order to evaluate the multivariate trigonometric polynomial

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in I_N^d} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}}, \qquad j = 0, \ldots, M-1,$$

at nonequispaced nodes $\boldsymbol{x}_j \in \mathbb{T}^d$. In this case, the final approximation (5) has $(2m+1)^d$ summands and the size of the FFT is n^d. The window function is of product type and factorizing the Gaussian window function asks for the storage or evaluation of $2dM$ numbers. The final cost of the multivariate NFFT is $\mathcal{O}\left(N^d \log N + |\log \varepsilon|^d M\right)$, where ε is the desired accuracy, see [9] for details. We summarize our findings in the following algorithm.

3 GPU Implementation

Programmable graphics processing units (GPUs) have evolved into highly parallel, multithreaded, manycore processors with enormous computational capacity and very high memory bandwidth. On these devices, a widely available programming language is the Compute Unified Device Architecture (CUDA) [16]. CUDA provides a C-like programming environment with extensions to differentiate between code and data structures meant for the host (central processing unit, CPU) and device execution. The host is responsible for initializing the device, transferring data between systems and device memory, and initiating execution of device kernels, see also Figure 1. In the following, we introduce a parallel nonequispaced FFT based on the sequential algorithm implemented in the NFFT3 library [12]. Because of the concept of graphics processors as coprocessors the original algorithm is inefficient on GPUs and we have to adapt some algorithmic parts as well as data structure to get a significant speedup.

Input: $d, M, N \in \mathbb{N}$, $\boldsymbol{x}_j \in [-\frac{1}{2}, \frac{1}{2})^d$, $j = 0, \ldots, M-1$, and $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_N^d$.

1: For $\boldsymbol{k} \in I_N^d$ compute $\hat{g}_{\boldsymbol{k}} = n^{-d} \cdot \hat{f}_{\boldsymbol{k}} / \hat{\varphi}(\boldsymbol{k})$. Roll-off correction (ROC)

2: For $\boldsymbol{l} \in I_n^d$ compute $g_{\boldsymbol{l}} = \sum_{\boldsymbol{k} \in I_N^d} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{l} / n}$. Fast Fourier transform (FFT)

3: For $j = 0, \ldots, M-1$ compute $f_j = \sum_{\boldsymbol{l} \in I_{n,m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}} \, \tilde{\varphi}(\boldsymbol{x}_j - \boldsymbol{l}/n)$. Convolution step (CONV)

Output: Approximate function values f_j, $j = 0, \ldots, M-1$.
Arithmetic cost: $\mathcal{O}(N^d \log(N) + \varepsilon^d M)$ + evaluations of the window function.

Algorithm 1: Nonequispaced fast Fourier transform (NFFT).

As described in the previous section, the NFFT consists of three steps which are implemented as kernel functions on the GPU. The first step of Algorithm 1, i.e. the roll-off correction, is straightforward to parallelize and we implement our kernel function like the serial algorithm without precomputation of $\hat{\varphi}$. Every thread (computational unit on GPU) computes on 2^d frequency indices $\boldsymbol{k} \in I_N^d$, realizes also the fftshift, and threads in one warp use a coalesced memory access. The second step is the fast Fourier transform of total size length n^d, available in the CUFFT library [17]. For interesting problem sizes, the convolution step is most time consuming step and the NFFT algorithm on the CPU is bounded by its runtime. Now, every thread computes the sum of step three of Algorithm 1 on one spatial index $j = 0, \ldots, M-1$ of the vector $\boldsymbol{f}$. We start by computing $\tilde{\varphi}$ for the current index j using the precomputed parts of the factorization of the window function. Again, the access to the FFT-output g_l, $l \in I_n^d$, is optimized regarding to coalesced memory. Since the FFT-length n is free of choice, we restrict to the most relevant case that n is a power of two. In particular, we replace the if-statements for boundary tests by a cheap bit operation. More specifically, the modulo operation is realized by a bitwise 'and' operation with $n-1$. Further optimization strategies, like using texture memory or shared memory, do not show an improved performance for this application.

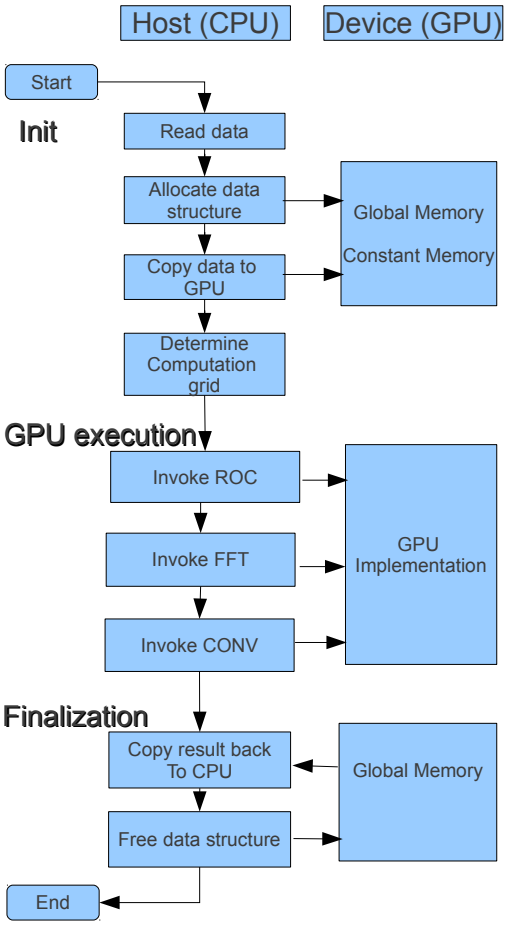

Fig. 1 Control and data flow of the CUNFFT.

4 Performance Comparison and Analysis

We test the algorithm in double precision arithmetic on a GEFORCE GTX 460 graphics card with 1GByte global memory. On the host system Intel(R) Core(TM) i5 CPU760@2.80GHz, we compile the NFFT3 library [12] by gcc 4.5.2 using the compiler option -O3.

Because of the bounded global memory on this graphics card, we have restrictions on the problem size. We assume a spatial dimension $d \in \mathbb{N}$, a fixed oversampling factor $\sigma = 2$, vectors $\hat{\boldsymbol{f}} \in \mathbb{C}^{N^d}$, $\boldsymbol{f} \in \mathbb{C}^M$, $\boldsymbol{x} \in \mathbb{R}^{dM}$ for the Fourier coefficients, samples, and nodes, and an inplace CUFFT on the vectors $\hat{\boldsymbol{g}}$, $\boldsymbol{g} \in \mathbb{C}^{n^d}$. Denoting by c and r, the size of complex and real floating numbers in bytes, respectively, yields

$$c(2(2N)^d + N^d + M) + dMr \leq 1\text{GByte}$$

and thus for $M = N^d$ the final restrictions in Table 1.

	$d=1$	$d=2$	$d=3$
double precision, $c=16$, $r=8$	$N = 2^{23}$	$N = 2^{11}$	$N = 2^7$
single precision, $c=8$, $r=4$	$N = 2^{24}$	$N = 2^{11}$	$N = 2^7$

Table 1 Restrictions on the problem size N of the CUNFFT.

Subsequently, we only consider the two-dimensional case $d = 2$. Performance measurements of the NFFT algorithm on CPU and GPU are shown in Figure 2. We choose $\sigma = 2$ and the cut-off parameter $m = 6$ as well as $M = N^2$ for all measurements corresponding to the default setting of the NFFT library on the CPU.

The speedup of the convolution step depends on the specific nodes $\boldsymbol{x}_j \in \mathbb{T}^2$, $j = 0, \ldots, M-1$. For clustered nodes, cf. Figure 2, we get a higher speedup as for uniformly distributed nodes $\boldsymbol{x}_j$, cf. Figure 3, because the global memory loads of g_l are more expensive for this case. The outlier for the ROC at $N^2 = 2^{16}$ is due to the behavior of the NFFT3 library.

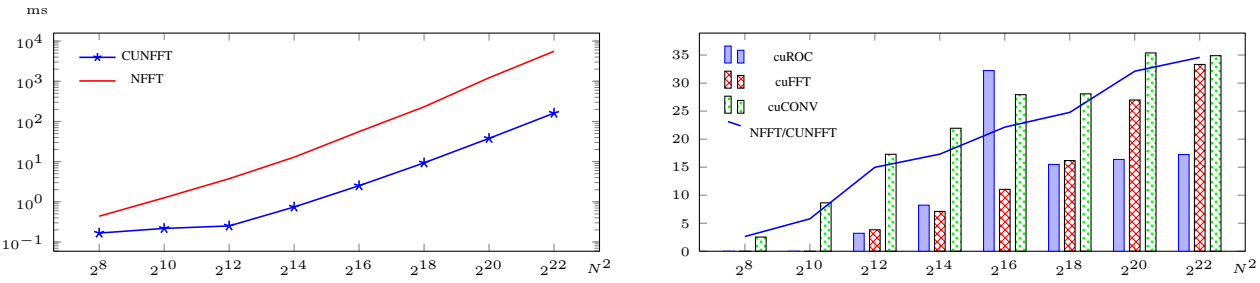

Fig. 2 Computing times in milliseconds (left) and speedup against NFFT3 library for clustered input nodes $\boldsymbol{x}_j$ (right).

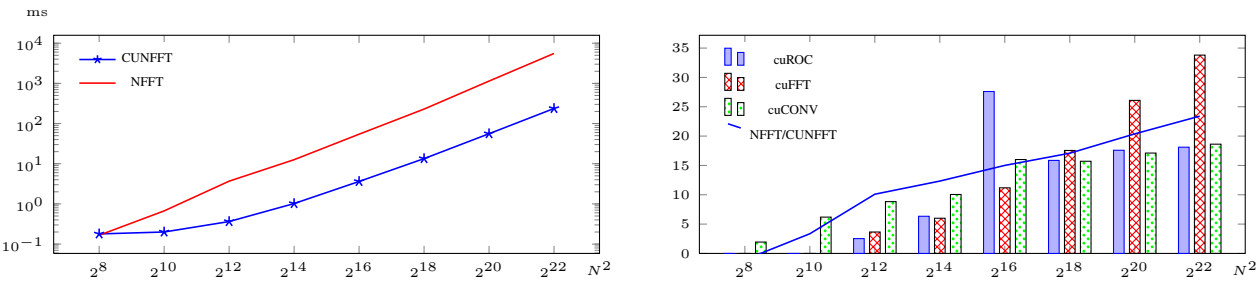

Fig. 3 Computing times in milliseconds (left) and speedup against NFFT3 library for uniformly distributed nodes $\boldsymbol{x}_j$ (right).

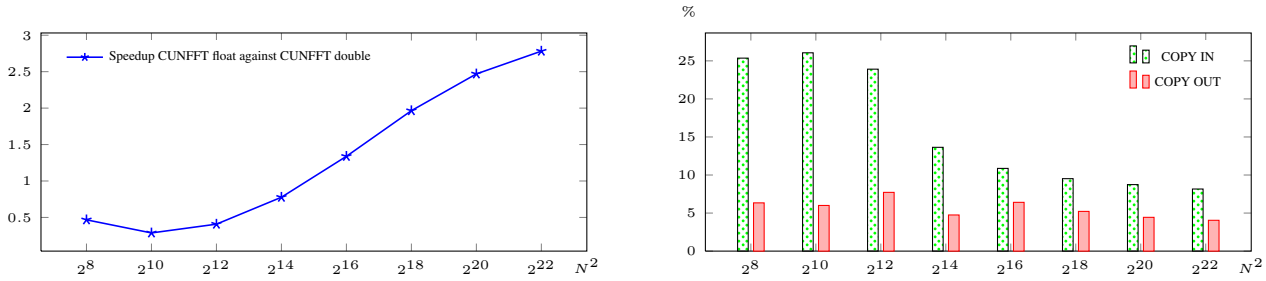

Fig. 4 Additional speedup for single precision (left) and data transfer times in comparison to the execution time of the CUNFFT (right).

Acknowledgment

The authors gratefully acknowledge support by the German Research Foundation within the project KU 2557/1-1 and by the Helmholtz Association within the young investigator group VH-NG-526.

References

[1] A. Dutt and V. Rokhlin, SIAM J. Sci. Comput. **14**(6), 1368–1393 (1993).
[2] A. Dutt and V. Rokhlin, Appl. Comput. Harmon. Anal. **2**(1), 85–100 (1995).
[3] G. Beylkin, Appl. Comput. Harmon. Anal. **2**(4), 363–381 (1995).
[4] G. Steidl, Adv. Comput. Math. **9**(3-4), 337–352 (1998).
[5] A. F. Ware, SIAM Rev. **40**, 838 – 856 (1998).
[6] D. Potts, G. Steidl, and M. Tasche, Fast Fourier transforms for nonequispaced data: A tutorial, in: Modern Sampling Theory: Mathematics and Applications, edited by J. J. Benedetto and P. J. S. G. Ferreira (Birkhäuser, Boston, MA, USA, 2001), chap. 12, pp. 247 – 270.
[7] J. A. Fessler and B. P. Sutton, IEEE Trans. Signal Process. **51**, 560 – 574 (2003).
[8] L. Greengard and J. Y. Lee, SIAM Rev. **46**(3), 443–454 (electronic) (2004).
[9] J. Keiner, S. Kunis, and D. Potts, ACM Trans. Math. Software **36**(4), Art. 19, 30 (2009).
[10] T. Sorensen, T. Schaeffter, K. Noe, and M. Hansen, IEEE Transactions on Medical Imaging **27**(4), 538 –547 (2008).
[11] P. Gwosdek, C. Schmaltz, J. Weickert, and T. Teubner, Preprint (2011).
[12] J. Keiner, S. Kunis, and D. Potts, NFFT 3.0, C subroutine library, `http://www.tu-chemnitz.de/~potts/nfft`.
[13] S. Kunis and S. Kunis, CUNFFT 1.0beta, CUDA subroutine library, `http://www.analysis.uni-osnabrueck.de`.
[14] C. F. Van Loan, Computational Frameworks for the Fast Fourier Transform (SIAM, Philadelphia, PA, USA, 1992).
[15] M. Frigo and S. G. Johnson, Proceedings of the IEEE **93**(2), 216 – 231 (2005).
[16] NVIDIA Corporation, NVIDIA CUDA C Programming Guide 2012, Version 4.2 , `http://developer.nvidia.com/nvidia-gpu-computing-documentation`.
[17] NVIDIA Corporation, CUDA Toolkit 4.1 CUFFT Library Programming Guide 2012 , `http://developer.nvidia.com/nvidia-gpu-computing-documentation`.