SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. C479–C504

© 2019 Society for Industrial and Applied Mathematics

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

A PARALLEL NONUNIFORM FAST FOURIER TRANSFORM LIBRARY BASED ON AN "EXPONENTIAL OF SEMICIRCLE" KERNEL[*]

ALEXANDER H. BARNETT[†], JEREMY MAGLAND[†], AND LUDVIG AF KLINTEBERG[‡]

Abstract. The nonuniform fast Fourier transform (NUFFT) generalizes the FFT to off-grid data. Its many applications include image reconstruction, data analysis, and the numerical solution of differential equations. We present FINUFFT, an efficient parallel library for type 1 (nonuniform to uniform), type 2 (uniform to nonuniform), or type 3 (nonuniform to nonuniform) transforms, in dimensions 1, 2, or 3. It uses minimal RAM, requires no precomputation or plan steps, and has a simple interface to several languages. We perform the expensive spreading/interpolation between nonuniform points and the fine grid via a simple new kernel—the "exponential of semicircle" $e^{\beta\sqrt{1-x^2}}$ in $x \in [-1,1]$—in a cache-aware load-balanced multithreaded implementation. The deconvolution step requires the Fourier transform of the kernel, for which we propose efficient numerical quadrature. For types 1 and 2, rigorous error bounds asymptotic in the kernel width approach the fastest known exponential rate, namely that of the Kaiser–Bessel kernel. We benchmark against several popular CPU-based libraries, showing favorable speed and memory footprint, especially in three dimensions when high accuracy and/or clustered point distributions are desired.

Key words. nonuniform, NFFT, spreading, kernel, Kaiser–Bessel, parallel

AMS subject classifications. 65T50, 65T40, 65Y05, 68N01

DOI. 10.1137/18M120885X

1. Introduction.

The need for fast algorithms for spectral analysis of nonuniformly sampled data arose soon after the popularization of the FFT in the 1960s. Many early methods came from signal processing [44] and astronomy [58, 39, 49], [50, sect. 13.8], but it was not until the 1990s that Dutt and Rokhlin [12] gave the first rigorous analysis of a convergent scheme. The nonuniform fast Fourier transform (NUFFT) has since become crucial in many areas of science and engineering. Several imaging methods, including magnetic resonance imaging (MRI) [57, 35], X-ray computed tomography (CT) [18], ultrasound diffraction tomography [9], and synthetic aperture radar [3], sample the Fourier transform at non-Cartesian points [60, 23] and hence require the NUFFT or its inverse for accurate reconstruction. Real-time Fourier-domain optical coherence tomography (OCT) relies on rapid one-dimensional (1D) NUFFTs [63]. Periodic electrostatic and Stokes problems are commonly solved by fast "particle-mesh Ewald" summation, whose spectral part is equivalent to a pair of NUFFTs [37, 42]. Spectrally accurate function interpolation may be efficiently performed with the NUFFT [30, sect. 6], [20]. The numerical approximation of Fourier transforms using non-Cartesian or adaptive quadrature grids arises in heat solvers [35], cryo-electron microscopy [64, 6], and electromagnetics [38]. Many more applications are found in the reviews [62, 33, 48, 30, 25].

Our purpose is to describe and benchmark a general-purpose software library for the NUFFT that achieves high efficiency with an open-source compiler by com-

[*] Submitted to the journal's Software and High-Performance Computing section August 21, 2018; accepted for publication (in revised form) June 10, 2019; published electronically September 19, 2019.
https://doi.org/10.1137/18M120885X

[†] Flatiron Institute, Simons Foundation, New York, NY 10010 (abarnett@flatironinstitute.org, jmagland@flatironinstitute.org).

[‡] Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (jafklint@sfu.ca).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

bining mathematical and implementation innovations. The computational task is to approximate, to a requested relative accuracy ε, the following exponential sums. Let $d = 1, 2$, or 3 be the spatial dimension. Let N_i be the number of desired Fourier modes in dimension $i = 1, \ldots, d$; in each dimension the Fourier mode (frequency) indices are

$$\mathcal{I}_{N_i} := \begin{cases} \{-N_i/2, \ldots, N_i/2 - 1\}, & N_i \text{ even,} \\ \{-(N_i-1)/2, \ldots, (N_i-1)/2\}, & N_i \text{ odd.} \end{cases}$$

The full set of mode indices is the Cartesian product that we denote by

$$\mathcal{I} = \mathcal{I}_{N_1,\ldots,N_d} := \mathcal{I}_{N_1} \times \cdots \times \mathcal{I}_{N_d} \ ,$$

containing a total number of modes $N = N_1 \cdots N_d$. The M nonuniform points have locations $\mathbf{x}_j$, $j = 1, \ldots, M$, which may be taken to lie in $[-\pi, \pi)^d$, with corresponding strengths $c_j \in \mathbb{C}$. Then the type 1 NUFFT (also known as the "adjoint NFFT" [48, 30]) approximates the outputs[1]

$$f_{\mathbf{k}} := \sum_{j=1}^{M} c_j e^{i\mathbf{k}\cdot\mathbf{x}_j} \ , \quad \mathbf{k} \in \mathcal{I} \qquad \text{(type 1, nonuniform to uniform).} \tag{1.1}$$

This may be interpreted as computing, for the 2π-periodic box, the N Fourier series coefficients of the distribution

$$f(\mathbf{x}) := \sum_{j=1}^{M} c_j \delta(\mathbf{x} - \mathbf{x}_j) \ . \tag{1.2}$$

Up to normalization, (1.1) generalizes the discrete Fourier transform (DFT), which is simply the uniform case, e.g., in one dimension, $\mathbf{x}_j = 2\pi j/M$ with $M = N_1$.

The type 2 transform (or "NFFT") is the adjoint of type 1. Unlike in the DFT case, it is not generally related to the inverse of type 1. It evaluates the Fourier series with given coefficients $f_{\mathbf{k}}$ at arbitrary target points $\mathbf{x}_j$, that is,

$$c_j := \sum_{\mathbf{k}\in\mathcal{I}} f_{\mathbf{k}} e^{-i\mathbf{k}\cdot\mathbf{x}_j} \ , \quad j = 1, \ldots, M \qquad \text{(type 2, uniform to nonuniform).} \tag{1.3}$$

Finally, the more general type 3 transform [35] (or "NNFFT" [30]) may be interpreted as evaluating the Fourier *transform* of the nonperiodic distribution (1.2) at arbitrary target frequencies $\mathbf{s}_k$ in $\mathbb{R}^d$, $k = 1, \ldots, N$, where k is a plain integer index, that is,

$$f_k := \sum_{j=1}^{M} c_j e^{i\mathbf{s}_k\cdot\mathbf{x}_j} \ , \quad k = 1, \ldots, N \qquad \text{(type 3, nonuniform to nonuniform).} \tag{1.4}$$

All three types of transform, (1.1), (1.3), and (1.4), consist simply of computing exponential sums that naively require $\mathcal{O}(NM)$ work. NUFFT algorithms compute these sums to a user-specified relative tolerance ε, in close to linear time in N and M.

Remark 1. In certain settings the above sums may be interpreted as quadrature formulae applied to evaluating a Fourier transform of a function. However, these tasks are not to be confused with the "inverse NUFFT" (see problems 4 and 5 in

[1] Note that our normalization differs from that of [12, 25].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[12, 25]) which involves, for instance, treating (1.3) as a linear system to be solved for $\{f_{\mathbf{k}}\}$, given the right-hand side $\{c_j\}$. For some nonuniform distributions this linear system can be very ill-conditioned. This inverse NUFFT is common in Fourier imaging applications; a popular solution method is to use conjugate gradients to solve the preconditioned normal equations, exploiting repeated NUFFTs for the needed matrix-vector multiplications [16, 18, 23, 60], [53, sect. 3.3]. Thus, efficiency gains reported here would also accelerate this inversion method. See [13, 31] for other approaches. We will not explicitly address the inversion problem here.

1.1. Prior algorithms, kernels, and implementations. There are two main approaches to the fast approximation of the sums (1.1) or (1.3), both of which build upon the FFT: (1) interpolation between nonuniform points and an upsampled regular grid, combined with an upsampled FFT and correction in Fourier space [12, 7, 16, 25, 30]; or (2) interpolation to/from an N-point (i.e. , not upsampled) regular grid, combined with the N-point FFT. In the univariate (1D) case, there are several variants of the second approach: Dutt and Rokhlin [13] proposed spectral Lagrange interpolation (using the cotangent kernel applied via the fast multipole method), combined with a single FFT. Recently, Ruis-Antolín and Townsend [53] proposed a stable Chebyshev approximation in intervals centered about each uniform point, which needs an independent N-point FFT for each of the $\mathcal{O}(\log 1/\varepsilon)$ coefficients, but is embarrassingly parallelizable. This improves upon earlier work [2] using Taylor approximation that was numerically unstable without upsampling (see [33, Ex. 3.10] and [53]).

We now turn to the first, and most popular, of the two above approaches. For the type 1 and type 2 transforms one sets up a regular fine grid of $n = \sigma^d N$ points where the upsampling factor in each dimension, $\sigma > 1$, is a small constant (typically ≤ 2). Taking the type 1 as an example, there are three steps. Step 1 evaluates on the fine grid the convolution of (1.2) with a smooth kernel function ψ, whose support has width w fine grid points in each dimension (see Figure 3.1(a)). This "spreading" requires $w^d M$ kernel evaluations. Step 2 applies the FFT on the n-point grid, needing $\mathcal{O}(N \log N)$ work. Step 3 extracts the lowest N frequencies from the result, then divides by $\hat{\psi}$, the Fourier transform of the kernel, evaluated at each of these frequencies; this is called deconvolution or roll-off correction. There is a class of kernels, including all those we discuss below, whose analysis gives an error ε decreasing exponentially (up to weak algebraic prefactors) with w, hence one may choose $w \approx c|\log \varepsilon|$. Thus the total effort for the NUFFT is $\mathcal{O}(M|\log \varepsilon|^d + N \log N)$.

The choice of spreading kernel ψ has a fascinating history. A variety of kernels were originally used for "gridding" in the imaging community (e.g. , see [26, 48, 30]). The truncated Gaussian kernel (see Figure 1.1) was the first for which an exponential convergence rate with respect to w was shown [12]. This rate was improved by Steidl [56, 14]: fixing σ, for an optimally chosen Gaussian width parameter the error is $\varepsilon = \mathcal{O}(e^{-\frac{1}{2}\pi(1-(2\sigma-1)^{-1})w})$. Beylkin [7] proposed B-splines for ψ, with the estimate $\varepsilon = \mathcal{O}((2\sigma-1)^{-w})$. In both cases, it is clear that increasing σ improves the convergence rate; however, since the cost of the upsampled FFT grows at least like σ^d, a tradeoff arises. In practice, many studies have settled on $\sigma = 2$ for general use [26, 18, 16, 30, 25, 46]. For this choice, both the above Gaussian and B-spline rates imply that $|\log_{10} \varepsilon|$, the number of correct digits, is approximately $0.5w$. For instance, to achieve 12 digits, a spreading width $w = 24$ is needed [25, Remark 2].

However, Jackson et al. [26] realized that the criteria for a good kernel ψ are very similar to those for a good *window function* in digital signal processing (DSP). We

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

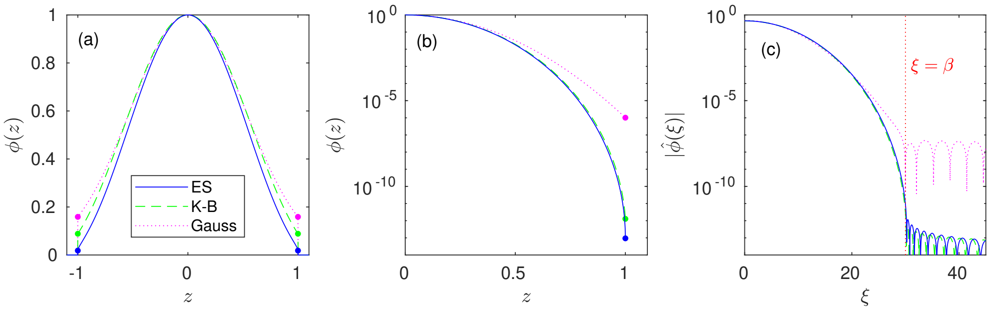

FIG. 1.1. *The proposed ES (exponential of semicircle) spreading kernel* (1.8) *(solid blue lines), the Kaiser–Bessel (KB) kernel* (1.5) *(dashed green), and the best truncated Gaussian (dotted pink)* $\phi(z) = e^{-0.46\beta z^2}$ *in* $|z| \le 1$. Figure (a) *shows all kernels for* $\beta = 4$. *The discontinuities at* ± 1 *are highlighted by dots.* Figure (b) *shows a logarithmic plot (for positive z) of the kernels for* $\beta = 30$ *(corresponding to a spreading width of* $w = 13$ *grid points). The graph for ES is a quarter-circle.* Figure (c) *shows the magnitude of the kernel Fourier transforms, and the "cutoff" frequency* $\xi = \beta$. *ES and KB have shape close to a quarter-ellipse in* $|\xi| < \beta$ *(see* (4.7) *and* (1.6)*). All have exponentially small values for* $|\xi| > \beta$*, but the Gaussian has exponential convergence rate in terms of* β *only around half that of ES or KB. (Figure in color online.)*

summarize these criteria:

(a) The numerical support of ψ in fine grid points, w, should be as small as possible, in order to reduce the $\mathcal{O}(w^d M)$ spreading cost.

(b) A certain norm of $\hat\psi(k)$ in the "tails" $|k| \ge (\sigma - \frac{1}{2})N$ should be as small as possible, relative to values in the central range $|k| < N/2$; see Figure 3.1(b).

The two criteria conflict: (a) states that ψ should be narrow, but (b), which derives from *aliasing error*, implies that ψ should be smooth. (We postpone the rigorous statement of (b) until (4.3).) It has been known since the work of Slepian and coworkers in the 1960s [54] that, if one chose L^2-norms in (b), the family of prolate spheroidal wavefunctions (PSWF) of order zero [45] would optimize the above criteria. It was also DSP folklore [28] that the "Kaiser–Bessel" (KB) kernel,

$$\phi_{\mathrm{KB},\beta}(z) := \begin{cases} I_0(\beta\sqrt{1-z^2})/I_0(\beta)\ , & |z| \le 1, \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

scaled here to have support $[-1, 1]$, where I_0 is the regular modified Bessel function of order zero [43, eq. (10.25.2)], well approximates the PSWF. However, unlike the PSWF, which is tricky to evaluate accurately [45], (1.5) needs only standard special function libraries [40]. Its Fourier transform (using the convention (3.1)) is known analytically[2] [28],

$$\hat\phi_{\mathrm{KB},\beta}(\xi) = \frac{2}{I_0(\beta)} \frac{\sinh\sqrt{\beta^2 - \xi^2}}{\sqrt{\beta^2 - \xi^2}}\ . \tag{1.6}$$

This transform pair (1.5)–(1.6) is plotted in green in Figure 1.1.

Starting with imaging applications in the 1990s, the KB kernel (1.5) was introduced for the NUFFT [26, 48, 16]. Note that the function (1.6), truncated to $[-\beta, \beta]$,

[2] This pair appears to be a discovery of B. F. Logan, and its use pioneered in DSP by J. F. Kaiser, both at Bell Labs, in the 1960s [21]. Curiously, the pair seems absent from all standard tables [22, section 6.677], [51, section 2.5.25], and [52, section 2.5.10].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

outside of which it is exponentially small, may instead be used as the spreading kernel [18, 30]. This latter approach—which we call "backward KB"[3]—has the computational advantage of spreading with cheaper sinh rather than I_0 evaluations. The error analyses of the two variants turn out to be equivalent. Despite being only conditionally convergent, the tail sum of (1.6) needed for criterion (b) may be bounded rigorously; this subtle analysis is due to Fourmont (see [17, pp. 30–38] and [18, section 4]). Its optimal convergence with w, summarized in [47, pp. 30–31] and [29, App. C], is (see (4.3) for the definition of the error ε_∞)

$$\varepsilon_\infty \;\le\; 4\pi(1-1/\sigma)^{1/4}\left(\sqrt{\frac{w-1}{2}}+\frac{w-1}{2}\right)e^{-\pi(w-1)\sqrt{1-1/\sigma}}\;. \tag{1.7}$$

This is the fastest known exponential error rate of any kernel, equalling that of the PSWF [4]: for the choice $\sigma = 2$ gives over $0.9w$ correct digits. This is *nearly twice* that of the Gaussian; 12 digits are reached with only $w = 13$. Attempts to further optimize this kernel give only marginal gains [16], unless restricted to cases with specific decay of the mode data $f_{\mathbf{k}}$ [41], or minimal upsampling ($\sigma \approx 1$) [27].

Turning to software implementations, most are based upon the Gaussian or KB kernels (in both its variants). Greengard and Lee [25] presented "fast Gaussian gridding" which reduced the number of exponential function evaluations from $w^d M$ to $(d+1)M$, resulting in a several-fold acceleration of the spreading step. This was implemented by those authors in a general-purpose single-threaded CMCL Fortran library [24]. The mature general-purpose NFFT code of Keiner, Kunis, and Potts [29, 30] is multithreaded [61] and uses backward KB by default (although fast Gaussian gridding is available). It allows various *precomputations* of kernel values (requiring a "plan" stage), demanding a larger RAM footprint but accelerating repeated calls with the same points. There are also several codes specialized to MRI, including MIRT (which uses full precomputation of the KB kernel) by Fessler [15], and recently BART [59] and PyNUFFT [36]. Various specialized GPU implementations also exist (reviewed in [46]), mostly for MRI [34, 32] or OCT [63]. Unlike general-purpose codes, these specialized packages tend to have limited accuracy or dimensionality, and tend not to document precisely what they compute.

1.2. Contribution of this work.

We present a general purpose documented CPU-based multithreaded C++ library (FINUFFT) [5] that is efficient without needing any precomputation stage. This means that the RAM overhead is very small and the interface simple. For medium and large problems in two dimensions and three dimensions its speed is competitive with state-of-the-art CPU-based codes. In some cases, at high accuracies, FINUFFT is faster than all known CPU-based codes by a factor of 10. The packages against which we benchmark are listed in Table 6.1.

We spread with a new "exponential of semicircle" (ES) kernel (see Figure 1.1),

$$\phi_\beta(z) := \begin{cases} e^{\beta(\sqrt{1-z^2}-1)}, & |z| \le 1 \\ 0 & \text{otherwise,} \end{cases} \tag{1.8}$$

which has error convergence rate arbitrarily close to that of (1.7); see Theorem 7. It is simpler and faster to evaluate than either of the KB pair (1.5)–(1.6), yet has essentially identical error. We demonstrate further acceleration via piecewise polynomial approximation. Equation (1.8) has no known analytic Fourier transform, yet

[3] The distinction between forward and backward use of the KB pair is unclear in the literature.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

we can use *numerical quadrature* to evaluate $\hat{\phi}_\beta$ when needed with negligible extra cost. Unlike interpolation from the fine grid (needed in type 2), spreading (needed for type 1) does not naturally parallelize over nonuniform points, because of collisions between threads writing to the output grid. However, we achieve efficiency in this case by adaptively blocking into auxiliary fine grids, after bin-sorting the points.

The rest of the paper is structured as follows. The next section outlines the software interfaces. In section 3 we describe the algorithms and parameter choices in full, including various novelties in terms of quadrature and type 3 optimization. In section 4 we summarize a rigorous aliasing error bound for the ES kernel, and use this to justify the choice of w and β. We also explain the gap between this bound and empirically observed relative errors. Section 6 compares the speed and accuracy performance against other libraries, in dimensions 1, 2, and 3. We conclude in section 7.

2. Use of the FINUFFT library.

The basic interfaces are very simple [5]. From C++, with `x` a `double` array of `M` source points, `c` a complex (`std::complex<double>`) array of `M` strengths, and `N` an integer number of desired output modes,

```
  status = finufft1d1(M,x,c,isign,tol,N,f,opts);
```

computes the 1D type 1 NUFFT with relative precision `tol` (see section 4.2), writing the answer into the preallocated complex array `f`, and returning zero if successful. Setting `isign` either 1 or -1 controls the sign of the imaginary unit in (1.1). `opts` is a struct defined by the required header `finufft.h` and initialized by `finufft_default_options(&opts)`, controlling various options. For example, setting `opts.debug=1` prints internal timings, whereas `opts.chkbnds=1` includes an initial check whether all points lie in the valid input range $[-3\pi, 3\pi]$. The above is one of nine routines with similar interfaces (types 1, 2, and 3 in dimensions 1, 2, and 3). The code is lightweight, at around 3300 lines of C++ (excluding interfaces to other languages). DFTs (discrete Fourier transforms) are performed by FFTW [19], which is the only dependency.

Interfaces from C and Fortran are similar to the above, and require linking with `-lstdc++`. From high-level languages one may call

```
  [f status] = finufft1d1(x,c,isign,tol,N,opts);    % MATLAB or octave,
  status = finufftpy.nufft1d1(x,c,isign,tol,N,f)    # python (numpy).
```

Here M is inferred from the input sizes. There also exists a Julia interface [1].

Remark 2. The above interface, since it does not involve any "plan" stage, incurs a penalty for repeated small problems (N and M of order 10^4 or less), traceable to the overhead (around 100 microseconds per thread in our tests) for calling `fftw_plan()` present when FFTW reuses stored wisdom. To provide maximal throughput for repeated small problems (which are yet not small enough that a dense matrix-matrix multiplication approach wins), we are adding interfaces that handle multiple inputs or allow a plan stage. At the time of writing these are available in two dimensions only, and will be extended in a future release.

3. Algorithms.

For type 1 we use the standard three-step procedure sketched above in section 1.1. For type 2 the steps are reversed. Type 3 involves a combination of types 1 and 2. Our Fourier transform convention is

$$\hat{\phi}(k) = \int_{-\infty}^{\infty} \phi(x) e^{ikx} dx \ , \qquad \phi(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{\phi}(k) e^{-ikx} dx \ . \tag{3.1}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

For the default upsampling factor $\sigma = 2$, given the requested relative tolerance ε, the kernel width w and ES parameter β in (1.8) are set via

$$w \;=\; \lceil \log_{10} 1/\varepsilon \rceil + 1\;, \qquad \beta \;=\; 2.30\, w\;. \tag{3.2}$$

The first formula may be summarized as follows: the kernel width is one more than the desired number of accurate digits. We justify these choices in section 4.2. (FINUFFT also provides a low-upsampling option $\sigma = 5/4$, which is not tested in this paper.)

3.1. Type 1: Nonuniform to uniform.

We describe the algorithm to compute $\tilde{f}_{\mathbf{k}}$, an approximation to the exact $f_{\mathbf{k}}$ defined by (1.1).

3.1.1. 1D case.

We use x_j to denote nonuniform source points, and $k \in \mathcal{I}$ to label the $N = N_1$ output modes. For FFT efficiency the DFT size n is chosen to be the smallest integer of the form $2^q 3^p 5^r$ not less than σN nor $2w$, the latter condition simplifying the spreading code.

Step 1 (spreading). From now on we abbreviate the ES kernel ϕ_β in (1.8) by ϕ. We rescale the kernel so that its support becomes $[-\alpha, \alpha]$, with

$$\alpha := wh/2 = \pi w/n\;, \tag{3.3}$$

where $h := 2\pi/n$ is the upsampled grid spacing. This rescaled kernel is denoted

$$\psi(x) := \phi(x/\alpha)\;, \qquad \text{thus} \quad \hat{\psi}(k) = \alpha\hat{\phi}(\alpha k) \qquad \text{(1D case)}, \tag{3.4}$$

and its periodization is

$$\tilde{\psi}(x) := \sum_{m\in\mathbb{Z}} \psi(x - 2\pi m) \qquad \text{(1D case)}. \tag{3.5}$$

We then compute, at a cost of wM kernel evaluations, the periodic discrete convolution

$$b_l = \sum_{j=1}^{M} c_j \tilde{\psi}(lh - x_j) \qquad \text{for } l = 0, \ldots, n-1\;, \tag{3.6}$$

as sketched in Figure 3.1(a).

Step 2 (FFT). We use the FFT to evalute the n-point DFT

$$\hat{b}_k = \sum_{l=0}^{n-1} e^{2\pi i l k/n} b_l \qquad \text{for } k \in \mathcal{I}\;. \tag{3.7}$$

Note that the output index set $\mathcal{I}$ is cyclically equivalent to the usual FFT index set $k = 0, \ldots, n-1$.

Step 3 (correction). We truncate to the central N frequencies, then diagonally scale (deconvolve) the amplitudes array, to give the outputs

$$\tilde{f}_k = p_k \hat{b}_k \qquad \text{for } k \in \mathcal{I}\;, \tag{3.8}$$

where a good choice of the correction factors p_k comes from samples of the kernel

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

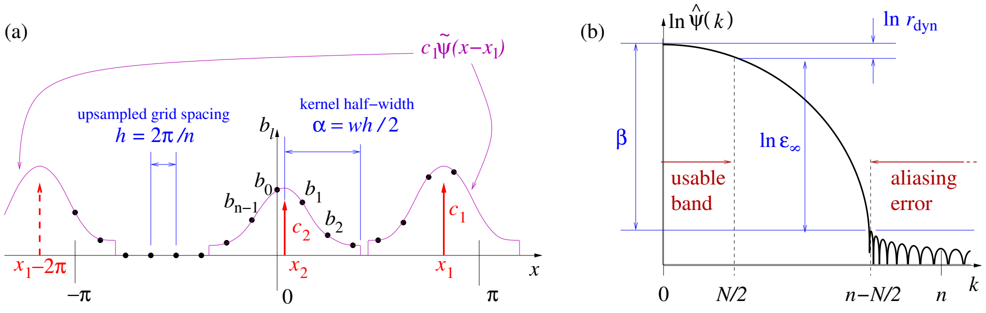

FIG. 3.1. *(a)* 1*D illustration of spreading from nonuniform points to the grid values* b_l*,* $l = 0,\dots,n-1$ *(shown as dots) needed for type* 1*. For clarity, only two nonuniform points* x_1 *and* x_2 *are shown; the former results in periodic wrapping of the effect of the kernel.* (b) *Semi-logarithmic plot of the (positive half of the) Fourier transform of the rescaled kernel* $\psi(x)$*, showing the usable frequency domain (and the dynamic range* r_{dyn} *over this domain), and a useful approximate relationship between the aliasing error bound* ε_∞ *and the ES kernel parameter* β*. From* (4.7)*, below cutoff the curve is well approximated by a quarter ellipse.*

Fourier transform,[4]

$$p_k = h/\hat{\psi}(k) \ = 2/(w\hat{\phi}(\alpha k)), \qquad k \in \mathcal{I}\ . \tag{3.9}$$

A new feature of our approach is that we approximate $\hat{\psi}(k)$ by applying Gauss–Legendre quadrature to the Fourier integral, as follows. This allows kernels without an analytically known Fourier transform to be used without loss of efficiency. Let q_j and w_j be the nodes and weights for a $2p$-node quadrature on $[-1,1]$. Since ϕ is real and even, only the p positive nodes are needed, thus

$$\hat{\psi}(k) \ = \ \int_{-\alpha}^{\alpha} \psi(x)e^{ikx}dx \ \approx \ wh\sum_{j=1}^{p} w_j\phi(q_j)\cos(\alpha k q_j)\ . \tag{3.10}$$

By a convergence study, we find that $p \geq 1.5w + 2$ (thus a maximum quadrature spacing close to h) gives errors less than ε, over the needed range $|k| \leq N/2$. A rigorous quadrature bound would be difficult due to the small square-root singularities at the endpoints in (1.8). The cost of the evaluation of p_k is $\mathcal{O}(pN)$, and naively would involve pN cosines. By exploiting the fact that, for each quadrature point q_j, successive values of $e^{i\alpha k q_j}$ over the regular k grid are related by a constant phase factor, these cosines can be replaced by only p complex exponentials, and pN adds and multiplies, giving an order of magnitude acceleration. We call this standard trick "phase winding."[5]

3.1.2. The case of higher-dimension $d > 1$. In general, different fine grid sizes are needed in each dimension. We use the same method, so that $n_i \geq \sigma N_i$, $n_i \geq 2w$, $n_i = 2^{q_i}3^{p_i}5^{r_i}$, $i = 1,\dots,d$. The kernel is a periodized product of scaled 1D kernels,

$$\psi(\mathbf{x}) = \phi(x_1/\alpha_1)\cdots\phi(x_d/\alpha_d)\ , \qquad \tilde{\psi}(\mathbf{x}) := \sum_{\mathbf{m}\in\mathbb{Z}^d} \psi(\mathbf{x}-2\pi\mathbf{m})\ , \tag{3.11}$$

[4] It is tempting instead to set p_k to be the DFT of the grid samples of the kernel $\{\tilde{\psi}(lh)\}_{l=0}^{n-1}$. However, in our experience this causes around twice the error of (3.9), as can be justified by the discussion in section 4. Fessler and Sutton [16, section V.C.3] report a similar finding.

[5] In the code, see the function `onedim_fseries_kernel` in `src/common.cpp`

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

where $\alpha_i = \pi w/n_i$. Writing $h_i := 2\pi/n_i$ for the fine grid spacing in each dimension, and $\mathbf{l} := (l_1, \ldots, l_d)$ to index each fine grid point, the discrete convolution becomes

$$b_{\mathbf{l}} = \sum_{j=1}^{M} c_j \tilde{\psi}((l_1 h_1, \ldots, l_d h_d) - \mathbf{x}_j) \ , \qquad l_i = 0, \ldots, n_i - 1, \quad i = 1, \ldots, d \ . \tag{3.12}$$

In evaluating (3.12), separability means that only wd kernel evaluations are needed per source point: the w^d square or cube of $\tilde{\psi}$ values is then filled by an outer product. The DFT (3.7) generalizes in the standard way to multiple dimensions. Finally, the correction factor is also separable,

$$p_{\mathbf{k}} = h_1 \ldots h_d \hat{\tilde{\psi}}(\mathbf{k})^{-1} = (2/w)^d (\hat{\phi}(\alpha_1 k_1) \cdots \hat{\phi}(\alpha_d k_d))^{-1} \ , \qquad \mathbf{k} \in \mathcal{I} \ , \tag{3.13}$$

so that only d repetitions of (3.10) are needed, followed by an outer product.

3.2. Type 2: Uniform to nonuniform.

To compute $\tilde{c}_j$, an approximation to c_j in (1.3), we reverse the steps for type 1. Given the number of modes N, and the precision ε, the choices of n, w, and β are as in type 1. From now on we stick to the case of general dimension d.

Step 1 (correction). The input coefficients $f_{\mathbf{k}}$ are precorrected (amplified) and zero-padded out to the size of the fine grid,

$$\hat{b}_{\mathbf{k}} = \begin{cases} p_{\mathbf{k}} f_{\mathbf{k}} \ , & \mathbf{k} \in \mathcal{I}, \\ 0 \ , & \mathbf{k} \in \mathcal{I}_{n_1,\ldots,n_d} \backslash \mathcal{I}, \end{cases} \tag{3.14}$$

with the same amplification factors $p_{\mathbf{k}}$ as in (3.13).

Step 2 (FFT). This is just as in type 1. Writing the general-dimension case of (3.7), with the index vectors $\mathbf{l}$ and $\mathbf{k}$ (and their ranges) swapped,

$$b_{\mathbf{l}} = \sum_{\mathbf{k} \in \mathcal{I}_{n_1,\ldots,n_d}} e^{2\pi i (l_1 k_1/n_1 + \cdots + l_d k_d/n_d)} \, \hat{b}_{\mathbf{k}} \qquad \text{for} \qquad l_i = 0, \ldots, n_i - 1, \quad i = 1, \ldots, d \ . \tag{3.15}$$

Step 3 (interpolation). The adjoint of spreading is interpolation, which outputs a weighted admixture of the grid values near to each target point. The output is then

$$\tilde{c}_j = \sum_{l_1=0}^{n_1-1} \cdots \sum_{l_d=0}^{n_d-1} b_{\mathbf{l}} \tilde{\psi}((l_1 h_1, \ldots, l_d h_d) - \mathbf{x}_j) \ . \tag{3.16}$$

As with type 1, because of separability, this requires wd evaluations of the kernel function, and w^d flops, per target point.

3.3. Type 3: Nonuniform to nonuniform.

This algorithm is more involved, but is a variant of standard ones (see [25, Alg. 3], [14, Alg. 2], [35], and [48, sect. 1.3]). Loosely speaking, it is "a type 1 wrapped around a type 2," where the type 2 replaces the middle FFT step of type 1. Given ε, we choose w, β, and p as before. We will present the choice of n_i shortly (see "step 0" below). It will involve the following bounds on source and target coordinates $\mathbf{x}_j = (x_j^{(i)}, \ldots, x_j^{(d)})$ and $\mathbf{s}_k = (s_k^{(i)}, \ldots, s_k^{(d)})$:

$$X_i := \max_{j=1,\ldots,M} |x_j^{(i)}| \ , \quad S_i := \max_{k=1,\ldots,N} |s_k^{(i)}| \qquad \text{for } i = 1, \ldots, d \ . \tag{3.17}$$

Step 1 (dilation and spreading). For spreading onto a grid on $[-\pi, \pi)^d$, a dilation factor γ_i needs to be chosen for each dimension $i = 1, \ldots, d$ such that the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

rescaled sources ${x'_j}^{(i)} := x_j^{(i)}/\gamma_i$ lie in $[-\pi,\pi)$. Furthermore, these rescaled coordinates must be at least $w/2$ grid points from the ends $\pm\pi$ in order to avoid wrap-around of mode amplitudes in step 2. This gives a condition relating n_i and γ_i,

$$X_i/\gamma_i \;\le\; \pi(1-w/n_i)\ , \qquad i=1,\ldots,d\ . \tag{3.18}$$

We may then rewrite (1.4) as $f_k := \sum_{j=1}^{M} c_j e^{i\mathbf{s}'_k\cdot\mathbf{x}'_j}$, $k=1,\ldots,N$, where ${s'_k}^{(i)} = \gamma_i s_k^{(i)}$.

We spread these rescaled sources $\mathbf{x}'_j = ({x'_j}^{(i)},\ldots,{x'_j}^{(d)})$ onto a regular grid using the usual periodized kernel (3.11) to get

$$\hat{b}_{\mathbf{l}} = \sum_{j=1}^{M} c_j\tilde{\psi}((l_1h_1,\ldots,l_dh_d)-\mathbf{x}'_j)\ , \qquad \mathbf{l}\in\mathcal{I}_{n_1,\ldots,n_d}\ . \tag{3.19}$$

Unlike before, here we have chosen a (cyclically equivalent) output index grid centered at the origin, because we shall now interpret $\mathbf{l}$ as a Fourier mode index.

Step 2 (Fourier series evaluation via type 2 NUFFT). Treating $\hat{b}_{\mathbf{l}}$ from (3.19) as Fourier series coefficients, we evaluate this series at rescaled target points using type 2 NUFFT (see section 3.2), thus

$$b_k = \sum_{\mathbf{l}\in\mathcal{I}_{n_1,\ldots,n_d}} \hat{b}_{\mathbf{l}}\, e^{i\mathbf{l}\cdot\mathbf{s}''_k} \qquad k=1,\ldots,N\ , \tag{3.20}$$

where the rescaled frequency targets have coordinates ${s''_k}^{(i)} := h_i {s'_k}^{(i)} = h_i\gamma_i s_k^{(i)}$, $i=1,\ldots,d$. Intuitively, the factor h_i arises because the fine grid of spacing h_i has to be stretched to unit spacing to be interpreted as a Fourier series.

Step 3 (correction). Finally, as in type 1, in order to compensate for the spreading in step 1 (in primed coordinates) a diagonal correction is needed,

$$\begin{aligned}&\tilde{f}_k = p_k b_k,\\ &p_k = h_1\ldots h_d\hat{\psi}(\mathbf{s}'_k)^{-1} = (2/w)^d\big(\hat{\phi}(\alpha_1{s'_k}^{(1)})\cdots\hat{\phi}(\alpha_d{s'_k}^{(d)})\big)^{-1}, \quad k=1,\ldots,N.\end{aligned}$$

But, in contrast to types 1 and 2, the set of frequencies at which $\hat{\phi}$ must be evaluated is *nonuniform*, so there is no phase winding trick. Rather, dpN cosines must be evaluated, recalling that p is the number of positive quadrature nodes. Despite this cost, this step consumes only a small fraction of the total computation time.

Remark 3. Empirically, we find that using the same overall requested precision ε in the above steps 1 and 2 gives overall error still close to ε. It has been shown in one dimension (see term E_3 in [14, p. 45]) that the type 3 error is bounded by the error in performing the above step 2 multiplied by $r_{\rm dyn}$, the dynamic range of $\hat{\psi}$ over the usable frequency band (see Figure 3.1(b)). Using $n\approx\sigma N$, (4.7), and (4.5) with $\gamma\approx 1$ we approximate

$$r_{\rm dyn} := \frac{\hat{\psi}(0)}{\hat{\psi}(N/2)} = \frac{\hat{\phi}(0)}{\hat{\phi}(\pi w/2\sigma)} \approx e^{\beta-\sqrt{\beta^2-(\pi w/2\sigma)^2}} = e^{\left(1-\sqrt{1-(2\sigma-1)^{-2}}\right)\beta}\ , \tag{3.21}$$

which for $\sigma=2$ gives $r_{\rm dyn}\approx e^{0.057\,\beta}$. From (3.2), $\beta\le 36$ for any $\varepsilon\ge 10^{-15}$, so $r_{\rm dyn}\le 8$, which is quite small. This helps to justify the above choice of tolerances.

Choice of fine grid size ("Step 0" for type 3). Finally, we are able to give the recipe for choosing the fine grid sizes n_i (which, of course, in practice precedes

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the above three steps). This relies on aliasing error estimates [14] for steps 1 and 3 that we explain here only heuristically. In section 4.1 we will see that spreading onto a uniform grid of size h_i induces a lattice of aliasing images separated by n_i in frequency space, so that the correction step is only accurate to precision ε out to frequency magnitude $n_i/2\sigma$. Thus, since $|\mathbf{s}'^{(i)}_k| \le \gamma_i S_i$ for all i and k, the condition

$$\gamma_i S_i \le \frac{n_i}{2\sigma} \ , \qquad i = 1, \dots, d, \tag{3.22}$$

is sufficient. Combining (3.18) and (3.22), then solving as equalities for the smallest n_i gives the recipe for the optimal parameters (similar to [35, Rem. 1]),

$$n_i \ = \ \frac{2\sigma}{\pi} X_i S_i + w \ , \qquad \gamma_i \ = \ \frac{n_i}{2\sigma S_i} \ , \qquad i = 1, \dots, d \ . \tag{3.23}$$

Remark 4 (FFT size for type 3). The product of the grid sizes n_i in each dimension $i = 1, \dots, d$ sets the number of modes, hence the FFT effort required, in the type 2 transform in step 2. Crucially, this is independent of the numbers of sources M and of targets N. Rather, n_i scales like the space-frequency product $X_i S_i$. This connects to the Fourier uncertainty principle: n_i scales as the number of "Heisenberg boxes" needed to fill the centered rectangle enclosing the data. In fact, since the number of degrees of freedom [55, p. 391] (or "semiclassical basis size" [11]) needed to represent functions living in the rectangle $[-X_i, X_i] \times [-S_i, S_i]$ is its area divided by 2π, namely $2X_i S_i/\pi$, we see that n_i is asymptotically σ times this basis size.

Efficiently handling poorly centered data. The above remark shows that type 3 is helped by translating all coordinates $\mathbf{x}_j$ and $\mathbf{s}_k$ so that their respective bounding boxes are centered around the origin. This reduces the bounds X_i and S_i defined by (3.17), hence reduces n_i and thus the cost of the FFT. Translations in $\mathbf{x}$ or in $\mathbf{s}$ are cheap to apply using the factorization

$$\sum_{j=1}^{M} c_j e^{i(\mathbf{s}_k+\mathbf{s}_0)\cdot(\mathbf{x}_j+\mathbf{x}_0)} \ = \ e^{i(\mathbf{s}_k+\mathbf{s}_0)\cdot\mathbf{x}_0} \sum_{j=1}^{M} (e^{i\mathbf{s}_0\cdot\mathbf{x}_j} c_j) e^{i\mathbf{s}_k\cdot\mathbf{x}_j} \ . \tag{3.24}$$

Thus the type 3 transform for translated data can be applied by prephasing the strengths by $e^{i\mathbf{s}_0\cdot\mathbf{x}_j}$, doing the transform, then postmultiplying by $e^{i(\mathbf{s}_k+\mathbf{s}_0)\cdot\mathbf{x}_0}$. The extra cost is $\mathcal{O}(N+M)$ complex exponentials. In our library, if input or output points are sufficiently poorly centered, we apply (3.24) using as $\mathbf{x}_0$ or $\mathbf{s}_0$ the means of the minimum and maximum coordinates in each dimension.

Remark 5 (type 3 efficiency). Remark 4 also shows that input data can be chosen for which the algorithm is arbitrarily inefficient. For example, with only two points ($M = N = 2$) in one dimension with $x_1 = -X$, $x_2 = X$, $s_1 = -S$, $s_2 = S$, then by choosing XS huge, (3.23) implies that the algorithm will require a huge amount of memory and time. Obviously, in such cases a direct summation of (1.4) is preferable. However, for N and M large but with clustered data, a butterfly-type algorithm which hierarchically exploits (3.24) could be designed; we leave this for future work.

4. Error analysis and parameter choices.

Here we summarize a rigorous estimate (proven in [4]) on the aliasing error of the 1D type 1 and 2 algorithms of section 3, when performed in exact arithmetic. We then use this to justify the algorithm parameter choices stated in (3.2). Finally, we evaluate and discuss the gap between this estimate and empirical errors.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

4.1. Theoretical results for the ES kernel. Let $\mathbf{f}$ be the vector of f_k outputs defined by (1.1) in one dimension, and let $\tilde{\mathbf{f}}$ be the analogous output of the above type 1 NUFFT algorithm in exact arithmetic. We use similar notation for type 2. By linearity, and the fact that the type 2 algorithm is the adjoint of type 1, the output aliasing error vectors must take the form

$$\tilde{\mathbf{f}} - \mathbf{f} = E\mathbf{c} \quad \text{(type 1)} , \qquad \tilde{\mathbf{c}} - \mathbf{c} = E^*\mathbf{f} \quad \text{(type 2)} \tag{4.1}$$

for some matrix E. Standard analysis (see [48, eq. (1.16)], [18, eq. (4.1)], [16, sect. V.B]), or [4]) involving the Poisson summation formula shows that, with the choice (3.9) for p_k, E has elements

$$E_{kj} = g_k(x_j) , \qquad \text{where} \qquad g_k(x) = \frac{1}{\hat{\psi}(k)} \sum_{m \neq 0} \hat{\psi}(k+mn) e^{i(k+mn)x} . \tag{4.2}$$

Since $|k| \le N/2$, error is thus controlled by a phased sum of the tails of $\hat{\psi}$ at frequency magnitudes at least $n - N/2$; see Figure 3.1(b).

It is usual in the literature to seek a uniform bound on elements of E by discarding the information about x_j, so that $|E_{kj}| \le \varepsilon_\infty \; \forall kj$, where

$$\varepsilon_\infty := \max_{|k| \le N/2} \|g_k\|_\infty \le \frac{\max_{|k|\le N/2,\, x \in \mathbb{R}} \left| \sum_{m\neq 0} \hat{\psi}(k+mn) e^{i(k+mn)x} \right|}{\min_{|k| \le N/2} |\hat{\psi}(k)|} . \tag{4.3}$$

The latter inequality is close to tight because r_{dyn}, defined by (3.21), controls the loss due to bounding numerator and denominator separately, and is in practice small.

Remark 6. A practical heuristic for ε_∞ is sketched in Figure 3.1(b): assuming that (i) $\hat{\psi}(k)$ decreases monotonically with $|k|$ for $|k| \le N/2$, and (ii) the worst-case sum (numerator in (4.3)) is dominated by the single value with smallest $|k|$, then we get $\varepsilon_\infty \approx |\hat{\psi}(n - N/2)/\hat{\psi}(N/2)|$, whose logarithm is shown in the figure.

A common use for (4.3) is the simple ℓ_1-ℓ_∞ bounds for (4.1) (see [56] and [16, p. 12]):

$$\max_{k \in \mathcal{I}} |\tilde{f}_k - f_k| \le \varepsilon_\infty \|\mathbf{c}\|_1 \quad \text{(type 1)}, \qquad \max_{1 \le j \le M} |\tilde{c}_j - c_j| \le \varepsilon_\infty \|\mathbf{f}\|_1 \quad \text{(type 2)}. \tag{4.4}$$

These results apply to any spreading kernel; we now specialize to the ES kernel. Fix an upsampling factor $\sigma > 1$. Given a kernel width w in sample points, one must choose in (1.8) an ES kernel parameter β such that $\hat{\psi}$ defined in (3.4) has decayed to its exponentially small region once the smallest aliased frequency $n - N/2 = n(1 - 1/2\sigma)$ is reached; see (4.3) and Figure 3.1(b). To this end we fix a "safety factor" γ, and set

$$\beta = \beta(w) := \gamma \pi w (1 - 1/2\sigma) , \tag{4.5}$$

so that for $\gamma = 1$ the exponential cutoff occurs exactly at $n - N/2$, while for $\gamma < 1$ the cutoff is safely smaller than $n - N/2$. With this set-up, the following states that the aliasing error converges almost exponentially with respect to the kernel width w.

Theorem 7 (see [4]). *For the* 1*D type* 1 *and* 2 *NUFFT, fix* N *and* σ *(hence the upsampled grid* $n = \sigma N$*) and the safety factor* $\gamma \in (0,1)$*. With* $\beta(w)$ *as in* (4.5)*, then the aliasing error uniform bound* (4.3) *converges with respect to kernel width* w *as*

$$\varepsilon_\infty = \mathcal{O}\left(\sqrt{w} e^{-\pi w \gamma \sqrt{1 - 1/\sigma - (\gamma^{-2} - 1)/4\sigma^2}} \right) , \qquad w \to \infty . \tag{4.6}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Its somewhat involved proof is detailed in [4]. A key ingredient is that, asymptotically as $\beta \to \infty$, $\hat{\phi}$ has the same "exponential of semicircle" form (up to algebraic factors) in the below-cutoff domain $(-\beta, \beta)$ that ϕ itself has in $(-1,1)$; compare Figures 1.1(b) and (c). Specifically, fixing the scaled frequency $\rho \in (-1,1)$, [4] proves that

$$\hat{\phi}(\rho\beta) \;=\; \sqrt{\frac{2\pi}{\beta}}\,\frac{1}{(1-\rho^2)^{3/4}}e^{\beta(\sqrt{1-\rho^2}-1)}\left[1+\mathcal{O}(\beta^{-1})\right]\;, \qquad \beta \to \infty\;. \tag{4.7}$$

Remark 8 (comparison to KB bounds). In the limit $\gamma \to 1^-$, (4.6) approaches the same exponential rate as (1.7), and with an algebraic prefactor improved by a factor $\sqrt{w}$. On the other hand, (1.7) has an explicit constant.

4.2. ES kernel parameter choices and empirical error. We now justify and test the parameter choices (3.2). With $\sigma = 2$, the factor 2.30 in (3.2) corresponds to a safety factor $\gamma \approx 0.976$ in (4.5), very close to 1. Note that $\gamma = 1$ would give a factor 2.356; however, we find that the factor 2.30 gives a slightly lower typical error for a given w than pushing γ closer or equal to 1 (this is likely due to the continued drop for $\xi > \beta$ visible in Figure 1.1(c)). Fessler and Sutton found a similar factor 2.34 when optimizing the KB kernel [16, Figure 11].

The width w is set by solving (4.6) with its algebraic prefactor dropped, to give $w \approx |\log \varepsilon_\infty|/\pi\gamma\sqrt{1-1/\sigma-(\gamma^{-2}-1)/4\sigma^2}+\text{const}$. Interpreting ε_∞ as the requested tolerance ε (see Remark 10 below), and inserting $\gamma \approx 0.976$, gives

$$w \;\approx\; 1.065|\log_{10}\varepsilon| \;+\; \text{const}\;. \tag{4.8}$$

As we show below, the factor 1.065 may be replaced by unity while still giving empirical errors close to the requested tolerance. The constant term in (4.8) is fit empirically. Thus we have explained the parameter choices (3.2).

In many applications one cares about relative ℓ_2 error in the output vector, which we will denote by

$$\epsilon := \frac{\|\tilde{\mathbf{f}}-\mathbf{f}\|_2}{\|\mathbf{f}\|_2} \quad \text{(types 1 and 3)}\;, \qquad \epsilon := \frac{\|\tilde{\mathbf{c}}-\mathbf{c}\|_2}{\|\mathbf{c}\|_2} \quad \text{(type 2)}\;. \tag{4.9}$$

Following many references [12, 48], we will use this metric for testing. Figure 4.1 measures this metric for FINUFFT for all nine transform types at two different problem sizes, with random data and randomly located nonuniform points. This shows that, using the choice (3.2), the achieved relative error ϵ well matches the requested tolerance ε, apart from when round-off error dominates. The mean slope of the logarithm of the empirical error ϵ with respect to that of ε in Figure 4.1 is slightly less than unity, due to the design choice of approximating the slope 1.065 in (4.8) by unity in (3.2).

Remark 9 (rounding error). Double-precision accuracy is used for all machine calculations in the library by default, and also in the studies in this work. The resulting rounding error is only apparent above aliasing error for the large 1D and two-dimensional (2D) transforms at high accuracy. Figure 4.1(a) shows that our library's accuracy is limited to nine relative digits in one dimension for $M = N = 10^7$; more generally, Figure 6.2 shows that rounding error is similar, and essentially the same for all tested libraries. When $M \approx N$ we find, in one dimension, that the rounding contribution to ϵ is roughly N times machine precision. Taking into account their

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

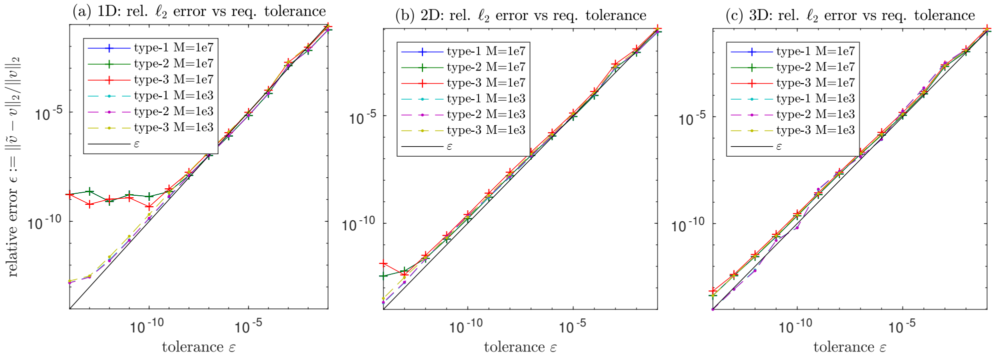

Fig. 4.1. *Comparison of empirical relative ℓ_2 error (ϵ) versus the requested tolerance (ε) for all nine FINUFFT routines. In each case $N \approx M$, with N_i roughly equal, with two problem sizes included ($M = 10^3$ and 10^7). Nonuniform points are uniformly randomly distributed in $[0, 2\pi)^d$, while strength data is complex Gaussian random. v denotes the true output vector, either $\mathbf{f}$ or $\mathbf{c}$ (this is computed with tolerance 10^{-15}), and $\tilde{v}$ the computed output vector at requested tolerance ε.*

choice of error norm, this concurs with the findings of [53, Figure 2.3]. See [48, section 1.4] for NUFFT rounding error analysis in one dimension. We observe in two dimensions and three dimensions that it is $\max_i N_i$ that scales the rounding error; thus, as Figure 4.1(b)–(c) shows, it is largely irrelevant even for large M.

Finally, we discuss the gap between any theoretical aliasing error estimate deriving from (4.3)—this includes (1.7) and (4.6)—and the *empirical* relative ℓ_2 error ϵ due to aliasing. The best possible type 1 bound from (4.1)–(4.3) is via the Frobenius norm $\|E\|_F \le \sqrt{MN}\varepsilon_\infty$, so

$$\|\tilde{\mathbf{f}} - \mathbf{f}\|_2 \;\le\; \sqrt{MN}\varepsilon_\infty \|\mathbf{c}\|_2.$$

Writing the transform (1.1) as $\mathbf{f} = A\mathbf{c}$, where A has elements $A_{kj} = e^{ikx_j}$, this gives

$$\epsilon \;\le\; \sqrt{MN}\varepsilon_\infty \frac{\|\mathbf{c}\|_2}{\|\mathbf{f}\|_2} \;\le\; \sqrt{MN}\varepsilon_\infty \frac{1}{\sigma_{\min}(A)}, \tag{4.10}$$

where in the last step the best bound applying to all nontrivial $\mathbf{c}$ is used, and $\sigma_{\min}(A)$ denotes the smallest singular value of A, or zero if $M > N$. Thus if $M > N$, there cannot be a general type 1 bound on ϵ, simply because, unlike the DFT, the *relative condition number* of the type 1 task (1.1) may be infinite (consider $\mathbf{c} \in \operatorname{Nul} A$, so $\mathbf{f} = \mathbf{0}$).[6]

However there are two distinct mechanisms by which (4.10) is pessimistic in real-world applications:

1. For *typical* input data, $\|\mathbf{f}\|_2$ is not smaller than $\|\mathbf{c}\|_2$; in fact (as would be expected from randomized phases in A), typically $\|\mathbf{f}\|_2 \approx \sqrt{N}\|\mathbf{c}\|_2$. The growth factor is close to $\sigma_{\max}(A)$. Thus the problem is generally well-conditioned. See Böttcher and Potts [8, section 4] for a formalization in terms of "probabilistic condition number."
2. The uniform bound (4.3) discards phase information in the elements of E, which, in practice, induce large cancellations to give errors that are improved

[6] The condition number may also be huge even if $M \le N$. The following MATLAB code, in which nonuniform points lie randomly in only *half* of the periodic interval, outputs typically 10^{-15}:
```
M=80; N=100; A = exp(1i*(-N/2:N/2-1)'*pi*rand(1,M)); min(svd(A))
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

by a factor $\sqrt{M}$ over bounds such as (4.4). Such ideas enable improved aliasing error estimates in a looser norm: e.g., interpreting (1.3) as point samples of a *function* $c(\mathbf{x}) = \sum_{\mathbf{k}\in\mathcal{I}} f_{\mathbf{k}} e^{-i\mathbf{k}\cdot\mathbf{x}}$, the error of the latter is easily bounded in $L^2([-\pi,\pi]^d)$, as clarified by Nestler [41, Lemma 1] (also see [27]).

Remark 10 (empirical ℓ_2 relative error). Assuming both mechanisms above apply in practice, they can be combined to replace (4.10) with the heuristic

$$\epsilon \approx \frac{\sqrt{N}\varepsilon_\infty \|\mathbf{c}\|_2}{\sqrt{N}\|\mathbf{c}\|_2} = \varepsilon_\infty ,$$

which justifies the interpretation of the requested tolerance ε as the uniform bound ε_∞ in (4.3) when setting kernel parameters. The result is that, as in Figure 4.1, barring rounding error, relative error ϵ is almost always similar to the tolerance ε.

To summarize, rather than design the kernel parameters around the rigorous but highly pessimistic worst-case analysis (4.10), we (as others do) design for typical errors. Thus, before trusting the relative error, the user is recommended to check that for their input data the desired convergence with respect to ε has been achieved.

5. Implementation issues.

Here we describe the main software aspects that accelerated the library. The chief computational costs in any NUFFT call are the spreading (types 1 and 3) or interpolation (type 2), scaling as $\mathcal{O}(w^d M)$, and the FFT, scaling as $\mathcal{O}(N \log N)$. We use the multithreaded FFTW library for the latter, thus in this analysis we focus on spreading/interpolation. In comparison, the correction steps, as explained in section 3, are cheap: 1D evaluations of $\hat{\psi}$ are easily parallelized over the p quadrature nodes (or, for type 3, the frequency points) with OpenMP, and the correction and reshuffling of coefficients is memory-bound and so does not benefit from parallelization.

5.1. Bin sorting of nonuniform points for spreading/interpolation.

When N is large, the upsampled grid (with $\sigma^d N$ elements) is too large to fit in cache. Unordered reads/writes to RAM are very slow (hundreds of clock cycles), thus looping through the nonuniform points in an order which preserves locality in RAM uses cache well and speeds up spreading and interpolation, by a factor of typically 2–10, including the time to sort.[7] Each nonuniform point requires accessing a block extending $\pm w/2$ grid points in each dimension, so there is no need to sort to the nearest grid point. Thus we set up boxes of size 16 grid points in the fast (x) dimension, and, in two dimensions or three dimensions, size 4 in the slower (y and z) dimensions. These sizes are a compromise between empirical speed and additional RAM needed for the sort. Then we do an "incomplete" histogram sort: we first count the number of points in each box and use this to construct the breakpoints between bins, then write point indices lying in each box into that bin, finally reading off the indices in the box ordering (without sorting inside each bin). This bin sort is multithreaded, except for the low-density case $M < N/10$ where we find that the single-thread version is usually faster.

5.2. Parallel spreading.

The interpolation task (3.16) parallelizes well with OpenMP, even for highly nonuniform distributions. Each thread is assigned a subset of the points $\{\mathbf{x}_j\}$; for each point it reads a block of size w^d from the fine grid, does a

[7] This is illustrated by running `test/spreadtestnd 3 1e7 1e7 1e-12 x 0 1` where `x` is `0` (no sort) or `1` (sort). For interpolation (`dir=2`), we find a speed-up factor 6 on a Xeon, or 14 on an i7.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

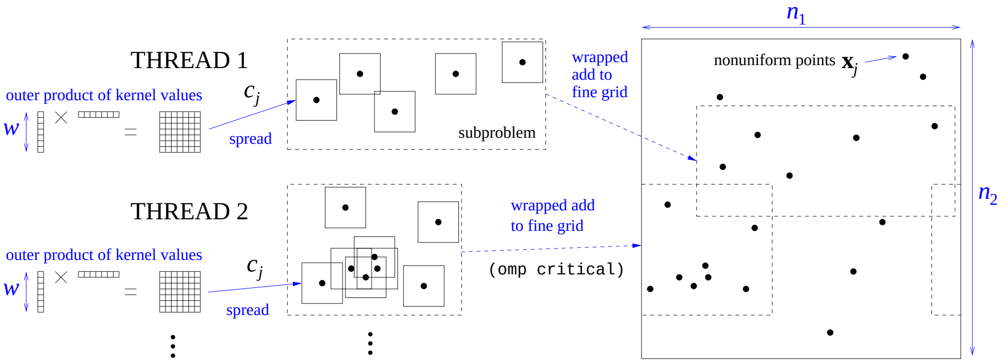

FIG. 5.1. *Sketch of parallel load balanced spreading scheme used for type* 1 *and type* 3 *transforms, showing the* 2D *case. Only two of the threads are shown.*

weighted sum using as weights the tensor product of 1D kernels, then writes the sum to a distinct output c_j.

In contrast, spreading (3.12) *adds* to blocks in the fine grid. Blocks being overwritten by different threads may collide, and, even if atomic operations are used to ensure a correct result, *false sharing* of cache lines [10, sect. 5.5.2] makes it inefficient. A conventional solution is to assign threads equal distinct slices of the fine grid [29]; however, for nonuniform distributions this could lead to arbitrarily poor load balancing. Instead, we group sorted points $\{\mathbf{x}_j\}$ into "subproblems" of size up to 10^4, which are assigned to threads; see Figure 5.1. (This choice is heuristic; an optimal size would depend on L3 cache and the number of threads.) To handle a subproblem, a thread finds the cuboid bounding all $\{\mathbf{x}_j\}$ in the subproblem, allocates a local cuboid of this size, spreads onto the cuboid, and finally adds the cuboid back into the fine grid. Since subproblems may overlap on the latter grid, this last step needs a `#pragma omp critical` block to avoid collisions between writes; however, this causes minimal overhead since almost all the time is spent spreading to cuboids. The scheme is adaptive: regardless of the point distribution, all threads are kept busy almost all of the time. The scheme requires additional RAM of order the fine grid size.

A further advantage is that the periodic wrapping of grid indices, which is slow, may be avoided: cuboids are padded by $w/2$ in each dimension and written to without wrapping. Index wrapping is only used when adding to the fine grid.

5.3. Piecewise polynomial kernel approximation. The 1D ES kernel (1.8) requires one real-valued `exp` and `sqrt` per evaluation. However, we find that the throughput depends drastically on the CPU type (i7 or Xeon), compiler (GCC versus Intel ICC), and kernel width w (the inner loop length that the compiler may be able to vectorize via SIMD instructions). For instance, a Xeon E5-2643 (with AVX2) with GCC version $\leq$ 7.x achieves only 40M evals/sec/thread, while the same CPU with ICC gives 50–200M evals/sec/thread. We believe this is due to compiler-provided `exp` instructions that exploit SIMD. Similar variations occur for the i7. Seeking a reliably efficient kernel evaluation on open-source compilers (e.g., GCC), we replaced the kernel evaluation by a polynomial look-up table. The result gives 350–600M evals/sec/thread, and accelerates 1D spreading/interpolation by a factor 2–3 (the effect in $d = 2, 3$ is less dramatic).

The look-up table works as follows. For each nonuniform point coordinate, the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

1D kernel $\psi(x)$ must be evaluated at w ordinates $x, x+h, \ldots, x+(w-1)h$ (see Figure 3.1(a)). We break the function ψ into w equal-width intervals and approximate each by a centered polynomial of degree p. Ordinates within each of the w intervals are then the same, allowing for SIMD vectorization. The approximation error need only be small relative to ε_∞; we find $p = w + 3$ suffices. Monomial coefficients are found by solving a Vandermonde system on collocation points on the boundary of a square in the complex plane tightly enclosing the interval. For each of the two available upsampling factors ($\sigma = 2$ and $5/4$), and all relevant w, we automatically generate C code containing all coefficients and Horner's evaluation scheme.

Remark 11. Piecewise polynomial approximation could confer on any kernel (e.g., KB, PSWF) this same high evaluation speed. However, AVX512 and future SIMD instruction sets may accelerate `exp` evaluations, making ES even faster. Since we do not know which will win with future CPUs, our library uses the ES kernel.

6. Performance tests.

Tests were run on a desktop with two Intel Xeon 3.4 GHz E5-2643 CPUs (each with 20 MB L3 cache), giving 12 total physical cores (up to 24 threads), and 128 GB RAM, running EL7 linux. Unless specified we compile all codes with GCC v.7.3.0. We compiled FINUFFT version 1.0 with flags `-fPIC -Ofast -funroll-loops -march=native`. Experiments were driven using the MEX interface to MATLAB R2016b. In the codes that use FFTW (i.e., FINUFFT and NFFT), we use version 3.3.3 and set its plan method to `FFTW_MEASURE` (see [19]), which sometimes takes a very long time to plan during the first call. Thus, to show realistic throughput we time only subsequent calls, for which FFTW looks up its stored plan. To minimize variation we take the best of the three subsequent calls.

Tasks. To assess the efficiency of our contributions—rather than merely measure the speed of FFTW—we choose "high density" tasks where M is somewhat larger than N, so the FFT is at most a small fraction of the total time. In one dimension, since timings do not vary much with point distribution, we always test with x_j i.i.d. uniform random in $[-\pi, \pi]$. For $d = 2, 3$ we use the following distributions (see insets in Figures 6.3–6.5):

- 2D "disc quad": a polar grid over the disc of radius π, using roughly $\sqrt{M}$ radial Gauss–Legendre nodes and $\sqrt{M}$ equispaced angular nodes.
- Three-dimensional (3D) "rand cube": i.i.d. uniform random in $[-\pi, \pi]^3$.
- 3D "sph quad": a spherical grid in the ball of radius π, using $\sqrt{M}/2$ radial Gauss–Legendre nodes and a $\sqrt{M} \times 2\sqrt{M}$ tensor-product grid on each sphere.

Here the first and last are realistic quadrature schemes for NUFFT applications [6]. They involve a divergence in point density at the origin of the form $r^{1/2-d}$ for $d = 2, 3$. We choose input strengths or coefficients as i.i.d. complex Gaussian random numbers.

Parallel scaling. Figure 6.1 shows parallel scaling tests of 3D type 1 and 2 FINUFFT. The highly nonuniform "sph quad" distribution was used in order to test the load balancing described in section 5.2. For $\varepsilon = 10^{-12}$, where each point interacts with $13^3 = 2197$ fine grid points, weak scaling (where M grows with p the number of threads) shows 90% parallel efficiency for $p \leq 12$ (one thread per physical core). Above this, hyperthreading is used: as expected, although it provides a slight net speed boost, measured in threads its parallel efficiency falls far short of that for $p \leq 12$. Strong scaling (acceleration at fixed M) is a tougher test, dropping to 62–74% at $p = 12$.

For a low-accuracy test ($\varepsilon = 10^{-3}$), the kernel is narrower, touching only $4^3 = 64$ fine grid points, thus the RAM access pattern is more random relative to the number of flops. We believe the resulting lower parallel efficiencies are due to memory bandwidth,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

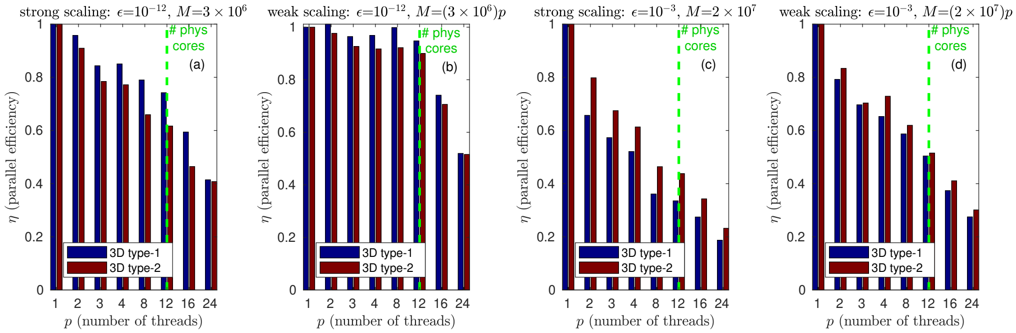

FIG. 6.1. *Parallel scaling of FINUFFT in 3D, for p threads of a Xeon desktop with 12 physical cores. Both type 1 and type 2 tasks are tested. In all cases there are $N = 100^3$ Fourier modes, and M, the number of "sph quad" nodes (see section 6), is as shown. Figures (a)–(b) show a high-accuracy case (12 digits). Figures (c)–(d) show low accuracy (three digits). For strong scaling the efficiency is the speed-up factor divided by p; for weak it is the speed-up factor for a problem size M proportional to p.*

TABLE 6.1
Summary of NUFFT libraries tested. "Kernel" lists the default spreading function ψ (some allow other kernels). "Language" is the coding language. A "yes" in the column "on-the-fly" indicates that no precomputation/plan phase is needed, hence low RAM use per nonuniform point. "omp" shows if multithreading (e.g., via OpenMP) is available. See sections 1.1 and 6.1 for more details.

Code name	Kernel	Language	On-the-fly	OMP	Periodic domain	Notes
FINUFFT	ES	C++	yes	yes	$[-\pi,\pi]^d$	
CMCL	Gaussian	Fortran	yes	no	$[-\pi,\pi]^d$	
NFFT	bkwd. KB	C	yes or no	yes	$[-1/2,1/2]^d$	
MIRT	optim. KB	MATLAB	no	no	$[-\pi,\pi]^d$	
BART	KB, $w=3$	C	yes	yes	$\prod_{i=1}^d[-N_i/2,N_i/2]$	$d=3$ only

rather than flops, being the bottleneck. That said, at $p = 24$ threads (full hyperthreading), both transforms are still 5–7 times faster than for a single core.

6.1. Benchmarks against existing libraries. We now compare FINUFFT against several popular open-source CPU-based NUFFT libraries mentioned in section 1.1. Their properties (including their periodic domain conventions) are summarized in Table 6.1. We study speed versus accuracy for types 1 and 2 for $d = 1, 2, 3$, covering most applications. The machine, OS, and default compiler were as above. In multithreaded tests we set $p = 24$ (two threads per core). Each code provides a MATLAB interface, or is native MATLAB. For reproducibility, we now list their test parameters and set-up (also see https://github.com/ahbarnett/nufft-bench):

- FINUFFT, version 1.0. Compiler flags are as in the previous section. We tested tolerances $\varepsilon = 10^{-2}, 10^{-3}, \ldots, 10^{-12}$.
- CMCL NUFFT, version 1.3.3 [24]. This ships with MEX binaries dated 2014. It uses `dfftpack` for FFTs. For fairness, we recompiled the relevant `*.mexa64` binaries on the test machine using `gfortran` with flags `-fPIC -Ofast -funroll-loops -march=native`, and `mex`. We tested tolerances $10^{-1}, 10^{-2}, \ldots, 10^{-11}$.
- NFFT, version 3.3.2 [29]. A compiler error resulted with GCC 7.x, so we used GCC 6.4.0. We used the default (backward KB) kernel. We used the "guru" interface with FFT grid size set to the smallest power of two at least $2N_i$,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

where N_i is the number of modes in dimension i, following their examples. Since they increased speed, we set the flags `PRE_PHI_HUT`, `FFT_OUT_OF_PLACE`, `NFFT_OMP_BLOCKWISE_ADJOINT`, and `NFFT_SORT_NODES` (the latter is not part of the standard MATLAB interface). We tested three variants:

  - no kernel precomputation; kernel is evaluated on the fly (labeled "NFFT");
  - "pre": option `PRE_PSI` which precomputes wdM kernel values (with tensor products done on the fly);
  - "full pre": option `PRE_FULL_PSI` which precomputes and then looks up all $w^d M$ kernel values.

  We tested kernel parameters $m = 1, 2, \ldots, 6$ (kernel width is $w = 2m + 1$), apart from "full pre" where RAM constraints limited us to $m = 1, 2$.

- MIRT (Michigan Image Reconstruction Toolbox), no version number; however, the latest changes to the `nufft` directory were on Dec. 13, 2016 [15]. This native MATLAB code precomputes a sparse matrix with all kernel values. Its matrix-vector multiplication appears to be single-threaded, thus we place this library in the single-threaded category. We use oversampling $\sigma = 2.0$ and the default kernel `minmax:kb`, which appears to be close to KB (1.5). RAM constraints limited us to testing width parameters $J = 2, 4$ (equivalent to our w).
- BART (Berkeley Advanced Reconstruction Toolbox), version 0.4.02 [59]. This is a recent multithreaded C code for three dimensions only by Uecker, Lustig, and Ong, used in a recent comparison by Ou [46]. We compiled with `-O3 -ffast-math -funroll-loops -march=native`. The MATLAB interface writes to and reads from temporary files; however, for our problem sizes with the use of a local SSD drive this adds less than 10% to the runtime. BART did not ship with periodic wrapping of the spreading kernel; however, upon request[8] we received a patched code `src/noncart/grid.c`. It has fixed accuracy ($w = 3$ is fixed). We empirically find that a prefactor $\sqrt{N_1 N_2 N_3}/1.00211$ gives around 5-digit accuracy (without the strange factor 1.00211 it gives only three digits).

The above notes also illustrate some of the challenges in setting up fair comparisons.

We now discuss the results (Figures 6.2–6.5). In each case we chose $M \approx 10N$, for reasons discussed above. To be favorable to codes that require precomputation, precomputation times were not counted (hence the label "after pre" in the figures). As in section 4.2, ϵ denotes relative l_2 error, measured against a ground truth of FINUFFT with tolerance $\varepsilon = 10^{-14}$.

1D comparisons. Figure 6.2 compares single-thread codes (left plots), and then, for a larger task, multithreaded codes (right plots). For single-threaded, FINUFFT outperforms all libraries except MIRT, which exploits MATLAB's apparently efficient sparse matrix-vector multiplication. However, what is not shown is that the precomputation time for MIRT is around 100 *times longer* than the transform time. For multithreaded type 1, FINUFFT is around 1.5–2× faster than NFFT without precomputation, but around 2× slower than "NFFT pre." For type 2, FINUFFT and "NFFT pre" are similar, but, of course, this does not count the precomputation time (and higher RAM overhead) of "NFFT pre." As per Remark 9, all libraries bottom out at around 9–10 digits due to rounding error.

2D comparisons. Figure 6.3 shows similar comparisons in 2D, for a point distribution concentrating at the origin. Compared to other codes not needing pre-

[8] M. Uecker, private communication.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

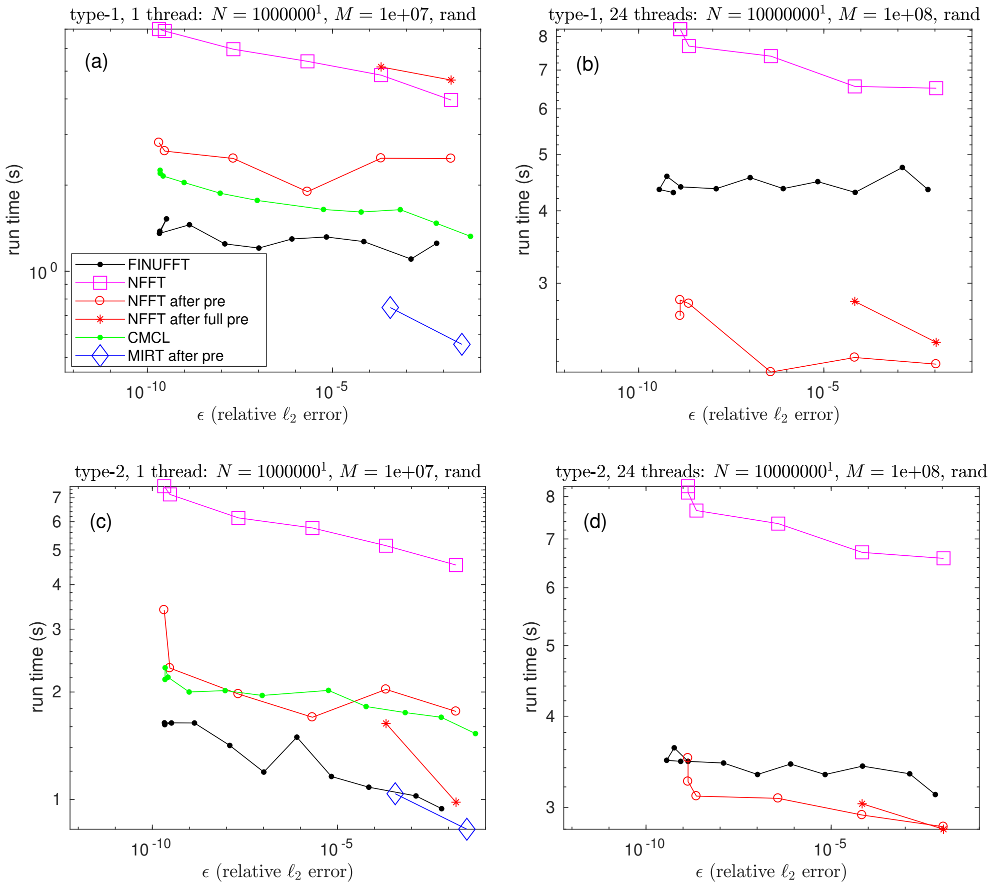

FIG. 6.2. *1D comparisons. Execution time versus accuracy is shown for various NUFFT libraries, for random data in 1D. Precomputations (needed for codes labeled "pre") were not included. The left shows single-threaded codes, the right multithreaded. The top pair are type* 1*, the bottom pair type* 2*. See section* 6.1.

computation, FINUFFT is 2–5× faster than CMCL (when single-threaded), and 4–8× faster than NFFT. When NFFT is allowed precomputation, its type 2 multithreaded speed is similar to FINUFFT, but for type 1 FINUFFT is 2× faster at high accuracy.

3D comparisons. Figure 6.4 compares single-threaded codes (now including BART). The left pair of plots shows random points: FINUFFT is at least 2× faster than any other code, apart from MIRT at 1-digit accuracy. CMCL is a factor 4–50× slower than single-threaded FINUFFT, we believe in part due to its lack of sorting nonuniform points. (The evidence is that for the right pair, where points have an ordered access pattern, CMCL is only 2–10× slower). NFFT without precomputation is 3–5× slower than FINUFFT; precomputation brings this down to 2–4×. As for $d = 1, 2$, we observe that "NFFT full pre" is no faster than "NFFT pre," despite its longer precomputation and larger RAM overhead.

Figure 6.5 shows larger multithreaded comparisons against NFFT; now we cannot include "NFFT full pre" due to its large RAM usage. At low accuracies with random points, FINUFFT and "NFFT pre" have similar speeds. However, for type 1 "sph

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

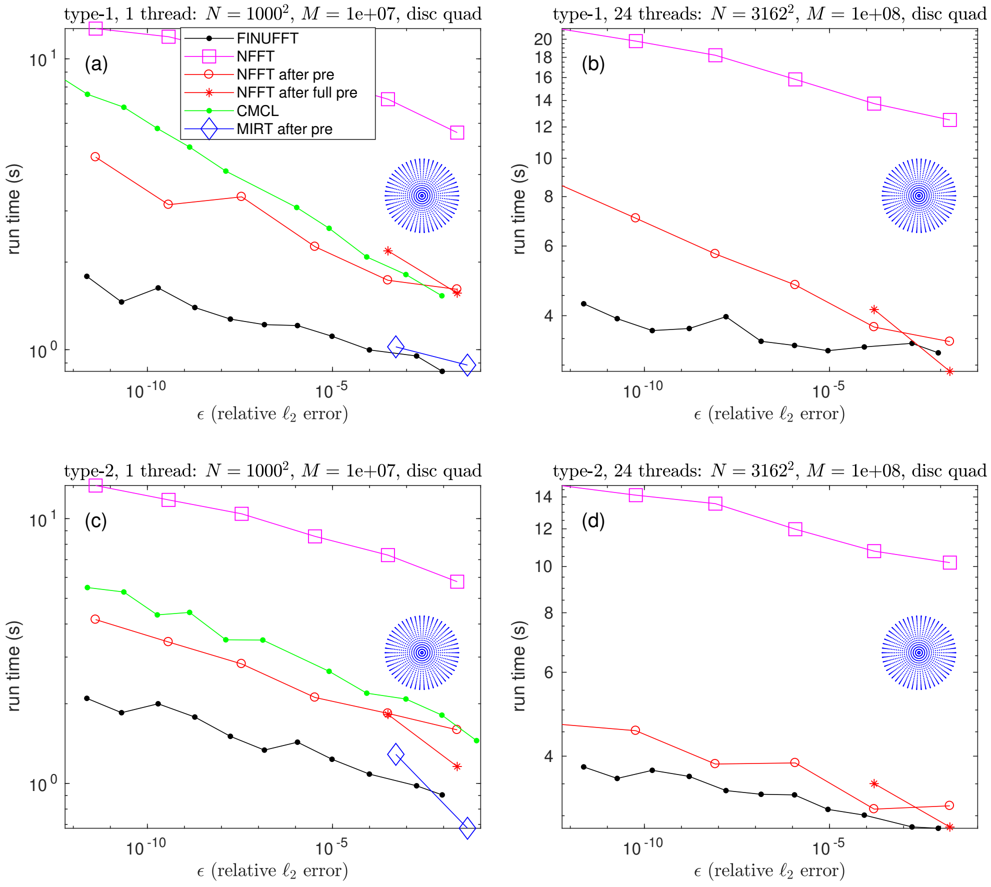

FIG. 6.3. *2D comparisons. Execution time versus accuracy is shown for the tested libraries, for 2D polar "disc quad" nodes (see section 6) illustrated in the insets at smaller M. Precomputations (needed for codes labeled "pre") were not included. The left figures show single-threaded codes, the right figures show multithreaded. The top pair are type 1, the bottom pair type 2. See section* 6.1.

quad" (panel (b)), for $\epsilon < 10^{-6}$, FINUFFT is 8–10× faster than NFFT even with precomputation. FINUFFT is at least 10× faster than BART in all cases.

In Table 6.2 we compare FINUFFT and NFFT in terms of both speed and memory overhead, for the same large, medium-accuracy, multithreaded task. We emphasize that, since $N_i = 256$, $i = 1, 2, 3$, is a power of two, the fine grids chosen by the two libraries are an identical $n_i = 512$. This means that the memory use, and the FFTW calls, are identical. Furthermore, the kernel widths are both $w = 7$ so the numbers of fine grid points written to are identical. If precomputations are excluded, FINUFFT is 16× faster than NFFT, and 8.6× faster than "NFFT pre." For a single use (i.e., including initialization and precomputation), these ratios become 17× and 13×, respectively.

Remark 12. We believe that the following explains the large type 1 performance gain of FINUFFT over NFFT for the 3D clustered "sph quad" nodes, shown by Figure 6.5(b) and Table 6.2. NFFT assigns threads to equal slices of the fine grid (in the x-direction), whereas FINUFFT uses subproblems which load balance regardless of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

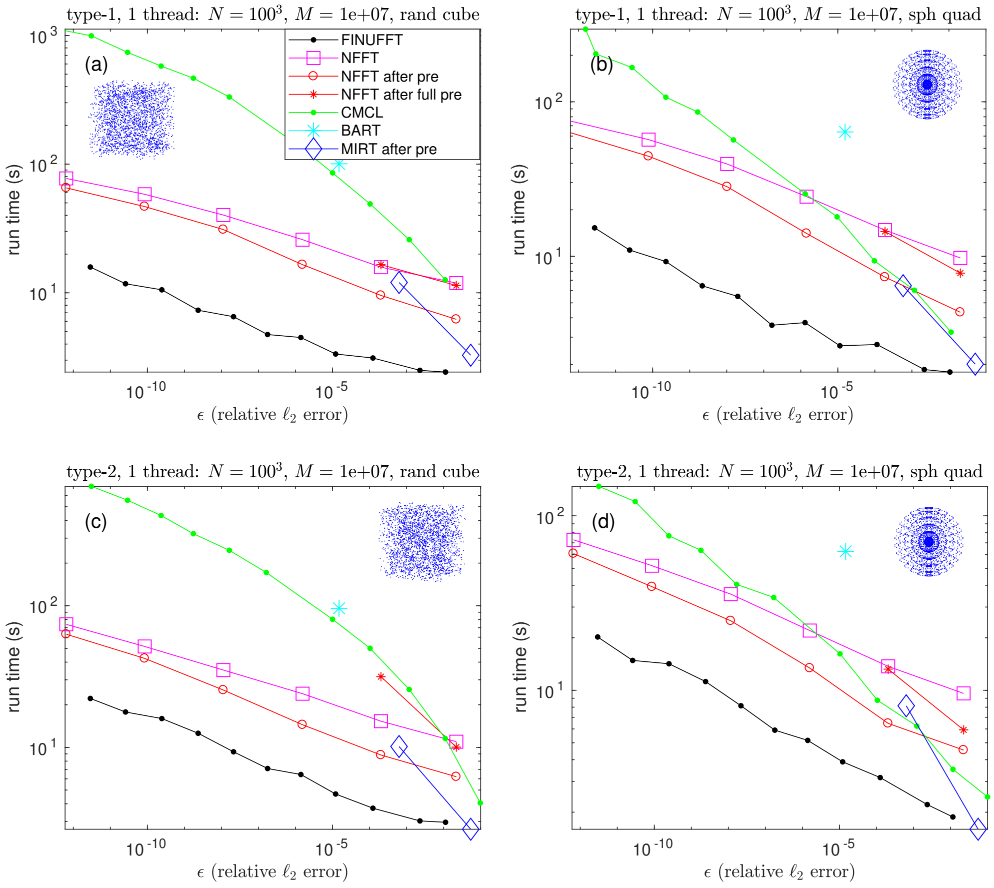

FIG. 6.4. *3D single-threaded comparisons. Execution time versus accuracy is shown for the tested libraries, for uniform random (left pair) and spherical quadrature (right pair) nodes (see section* 6*). Node patterns are shown in the insets at a smaller M. Precomputations (needed for codes labeled "pre") were not included. The top pair are type* 1*, the bottom pair type* 2*. See section* 6.1.

TABLE 6.2

Performance of FINUFFT and NFFT for a large 3D type 1 *transform with roughly* 6*-digit accuracy, using* 24 *threads. $N = 256^3$ modes are requested with $M = 3 \times 10^8$ "sph quad" nonuniform points. The spreading time dominates over the FFT. For NFFT, $t_{\text{plan+pre}}$ counts both planning and kernel precomputation. RAM is measured using* `top`*, relative to the baseline (around* 12 *GB) needed to store the input data in MATLAB. See section* 6 *for machine and NFFT parameters.*

Code and parameters	ϵ (rel. ℓ_2 error)	$t_{\text{plan+pre}}$	t_{run}	RAM overhead
FINUFFT (tol. 10^{-6})	1.4e-06	N/A	14.6 s	8.8 GB
NFFT ($m = 3$)	4.7e-06	10.4 s	238 s	20.9 GB
NFFT ($m = 3$) `PRE_PSI`	4.7e-06	67.3 s	125 s	67.1 GB

the clustering of nodes. We find that only a couple of threads are active for the entire run time of NFFT; most complete their jobs quickly, giving low parallel efficiency.

Finally, Table 6.2 shows that if NFFT precomputation is used, its RAM overhead is around 8× that of FINUFFT. The "NFFT pre" RAM overhead of around 28

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

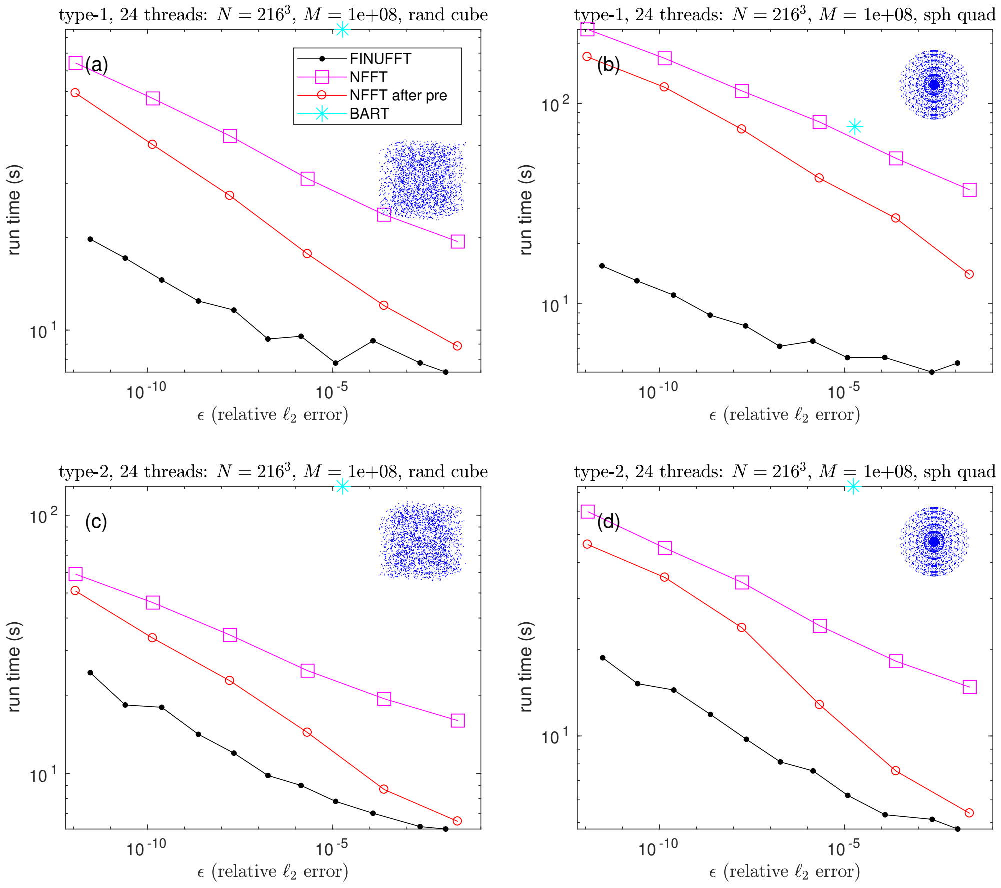

FIG. 6.5. *3D multithreaded comparisons. For an explanation, see the caption of Figure* 6.4.

`double`s per point is consistent with the expected $wd = 21$ stored kernel values per point.

7. Conclusion. We have presented an open-source parallel CPU-based general-purpose type 1, 2, and 3 NUFFT library (FINUFFT) for dimensions 1, 2, and 3 that is competitive with existing CPU-based libraries. Using a new spreading kernel (1.8), all kernel evaluations are efficiently done on the fly: this avoids any precomputation phase, keeps the RAM overhead small, and allows for a simple user-friendly interface from multiple languages. Efficient parallelization balances the work of threads, adapting to any nonuniform point distribution. For all three types, we introduce numerical quadrature to evaluate a kernel Fourier transform for which there is no known analytic formula. Rigorous estimates show almost exponential convergence of the kernel aliasing error, with rate arbitrarily close to that of the best known. We explained the gap between such estimates and empirical relative errors. We benchmarked several NUFFT libraries in detail. We showed that for certain 3D problems with clustered distributions FINUFFT is an order of magnitude faster than the other libraries, even when they are allowed precomputation. In the latter case, FINUFFT has an order of magnitude less RAM overhead.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

There are several directions for future work, starting with benchmarking the less-common type 3 case. An efficient interface for the case of repeated small problems (Remark 2) should be completed. There is also a need for a carefully benchmarked general-purpose GPU NUFFT code, following Kunis and Kunis [34], Ou [46], and others. Implementation of both of the above is in progress.

Remark 13. In some particle-mesh Ewald applications [37] one needs spatial derivatives of the spreading kernel.[9] However, (1.8) has unbounded derivatives (with inverse square-root singularity) at the endpoints. Instead one may prefer the exponentially close variant $\phi(z) = 2e^{-\beta}\cosh\beta\sqrt{1-z^2}$ since it is smooth up to the endpoints. This kernel requires one extra reciprocal, or approximation as in section 5.3.

Acknowledgments. We are grateful for discussions with Joakim Andén, Charlie Epstein, Zydrunas Gimbutas, Leslie Greengard, Hannah Lawrence, Andras Pataki, Daniel Potts, Vladimir Rokhlin, Yu-hsuan Shih, Marina Spivak, David Stein, and Anna-Karin Tornberg. The Flatiron Institute is a division of the Simons Foundation.

REFERENCES

[1] L. af Klinteberg, *Julia Interface to FINUFFT*, 2018, https://github.com/ludvigak/FINUFFT.jl.

[2] C. Anderson and M. D. Dahleh, *Rapid computation of the discrete Fourier transform*, SIAM J. Sci. Comput., 17 (1996), pp. 913–919, https://doi.org/10.1137/0917059.

[3] F. Andersson, R. Moses, and F. Natterer, *Fast Fourier methods for synthetic aperture radar imaging*, IEEE Trans. Aerosp. Electron. Syst., 48 (2012), pp. 215–229.

[4] A. H. Barnett, *Asymptotic aliasing error of the kernel* $exp(\beta\sqrt{1-x^2})$ *in non-uniform fast Fourier transforms*, 2019, in preparation.

[5] A. H. Barnett, J. Magland, and L. af Klinterberg, *Flatiron Institute Nonuniform Fast Fourier Transform Libraries (FINUFFT)*, 2018, https://github.com/flatironinstitute/finufft.

[6] A. H. Barnett, M. Spivak, A. Pataki, and L. Greengard, *Rapid solution of the cryo-EM reconstruction problem by frequency marching*, SIAM J. Imaging Sci., 10 (2017), pp. 1170–1195, https://doi.org/10.1137/16M1097171.

[7] G. Beylkin, *On the fast Fourier transform of functions with singularities*, Appl. Comput. Harmon. Anal., 2 (1995), pp. 363–383.

[8] A. Böttcher and D. Potts, *Probability against condition number and sampling of multivariate trigonometric random polynomials*, Electron. Trans. Numer. Anal., 26 (2007), pp. 178–189.

[9] M. M. Bronstein, A. M. Bronstein, M. Zibulevsky, and H. Azhari, *Reconstruction in diffraction ultrasound tomography using nonuniform FFT*, IEEE Trans. Med. Imag., 21 (2002), pp. 1395–1401.

[10] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP. Portable Shared Memory Parallel Programming*, MIT Press, Cambridge, MA, 2008.

[11] M. J. Davis and E. J. Heller, *Semiclassical Gaussian basis set method for molecular vibrational wave functions*, J. Chem. Phys., 71 (1979), pp. 3383–3395.

[12] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14 (1993), pp. 1369–1393, https://doi.org/10.1137/0914081.

[13] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data,* II, Appl. Comput. Harmon. Anal., 2 (1995), pp. 85–100.

[14] B. Elbel and G. Steidl, *Fast Fourier transform for nonequispaced data*, in Approximation Theory IX, Vanderbilt University Press, Nashville, TN, 1998, pp. 39–46.

[15] J. Fessler, *Michigan Image Reconstruction Toolbox*, 2016, https://web.eecs.umich.edu/~fessler/irt/fessler.tgz.

[16] J. Fessler and B. Sutton, *Nonuniform fast Fourier transforms using min-max interpolation*, IEEE Trans. Signal Process., 51 (2003), pp. 560–574.

[17] K. Fourmont, *Schnelle Fourier-Transformation bei nichtäquidistanten Gittern und tomographische Anwendungen*, Ph.D. thesis, University of Münster, Münster, Germany, 1999.

[9] A.-K. Tornberg, personal communication.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[18] K. Fourmont, *Non-equispaced fast Fourier transforms with applications to tomography*, J. Fourier Anal. Appl., 9 (2003), pp. 431–450.

[19] M. Frigo and S. G. Johnson, *FFTW*, http://www.fftw.org/.

[20] Z. Gimbutas and S. Veerapaneni, *A fast algorithm for spherical grid rotations and its application to singular quadrature*, SIAM J. Sci. Comput., 5 (2013), pp. A2738–A2751, https://doi.org/10.1137/120900587.

[21] A. Goldstein and J. Abbate, *Oral-History: James Kaiser*, http://ethw.org/Oral-History:James_Kaiser, 1997, (accessed 2017-04-15).

[22] I. Gradshteyn and I. Ryzhik, *Table of Integrals, Series and Products*, 8th ed., Elsevier/Academic Press, Amsterdam, 2015.

[23] L. Greegard, J.-Y. Lee, and S. Inati, *The fast sinc transform and image reconstruction from nonuniform samples in k-space*, Commun. Appl. Math. Comput. Sci., 1 (2006), pp. 121–131.

[24] L. Greengard and J.-Y. Lee, *NUFFT libraries in Fortran*, https://www.cims.nyu.edu/cmcl/nufft/nufft.html, (accessed 2017-04-10).

[25] L. Greengard and J.-Y. Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Rev., 46 (2004), pp. 443–454, https://doi.org/10.1137/S003614450343200X.

[26] J. I. Jackson, C. H. Meyer, D. G. Nishimura, and A. Macovski, *Selection of a convolution function for Fourier inversion using gridding*, IEEE Trans. Med. Imag., 10 (1991), pp. 473–478.

[27] M. Jacob, *Optimized least-square nonuniform fast Fourier transform*, IEEE Trans. Signal Process., 57 (2009), pp. 2165–2177.

[28] J. Kaiser, *Digital filters*, in System Analysis by Digital Computer, J. Kaiser and F. Kuo, eds., John Wiley & Sons, New York, 1966, pp. 218–285.

[29] J. Keiner, S. Kunis, and D. Potts, *NFFT*, http://www-user.tu-chemnitz.de/~potts/nfft, 2002–2016, (accessed 2017-04-10).

[30] J. Keiner, S. Kunis, and D. Potts, *Using NFFT 3—A software library for various nonequispaced fast Fourier transforms*, ACM Trans. Math. Software, 36 (2009), 19.

[31] M. Kircheis and D. Potts, *Direct inversion of the nonequispaced fast Fourier transform*, Linear Algebra Appl., 575 (2019), pp. 106–140

[32] F. Knoll, A. Schwarzl, C. Diwoki, and D. K. Sodickson, *gpuNUFFT—An open-source GPU library for 3D gridding with direct MATLAB interface*, Proc. Intl. Soc. Mag. Reson. Med., 22 (2014), p. 4297; available online at https://github.com/andyschwarzl/gpuNUFFT.

[33] S. Kunis, *Nonequispaced FFT: Generalisation and Inversion*, Ph.D. thesis, Universität zu Lübeck, Lübeck, Germany, 2006.

[34] S. Kunis and S. Kunis, *The nonequispaced FFT on graphics processing units*, Proc. Appl. Math. Mech., 12 (2012), pp. 7–10.

[35] J.-Y. Lee and L. Greengard, *The type 3 nonuniform FFT and its applications*, J. Comput. Phys., 206 (2005), pp. 1–5.

[36] J.-M. Lin, *Python Non-Uniform Fast Fourier Transform (PyNUFFT): Multi-dimensional Non-Cartesian Image Reconstruction Package for Heterogeneous Platforms and Applications to MRI*, preprint, 2017, https://arxiv.org/abs/1710.03197.

[37] D. Lindbo and A.-K. Tornberg, *Spectral accuracy in fast Ewald-based methods for particle simulations*, J. Comput. Phys., 230 (2011), pp. 8744–8761.

[38] Q. H. Liu and N. Nguyen, *An accurate algorithm for nonuniform fast Fourier transforms (NUFFT's)*, IEEE Microw. Wirel. Compon. Lett., 8 (1998), pp. 18–20.

[39] D. D. Meisel, *Fourier transforms of data samples at unequal observational intervals*, Astronom. J., 83 (1978), pp. 538–545.

[40] S. L. Moshier, *CEPHES Mathematical Function Library*, 1984–1992, http://www.netlib.org/cephes.

[41] F. Nestler, *Automated parameter tuning based on RMS errors for nonequispaced FFTs*, Adv. Comput. Math., 42 (2016), pp. 889–919.

[42] F. Nestler, M. Pippig, and D. Potts, *Fast Ewald summation based on NFFT with mixed periodicity*, J. Comput. Phys., 285 (2015), pp. 280–315.

[43] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, eds., *NIST Handbook of Mathematical Functions*, Cambridge University Press, Cambridge, 2010, http://dlmf.nist.gov.

[44] A. Oppenheim, D. Johnson, and K. Steiglitz, *Computation of spectra with unequal resolution using the fast Fourier transform*, Proc. IEEE, 59 (1971), pp. 299–301.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/15/19 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[45] A. Osipov, V. Rokhlin, and H. Xiao, *Prolate Spheroidal Wave Functions of Order Zero: Mathematical Tools for Bandlimited Approximation*, Appl. Math. Sci. 187, Springer, New York, 2013.

[46] T. Ou, *gNUFFTW: Auto-Tuning for High-Performance GPU-Accelerated Non-Uniform Fast Fourier Transforms*, Technical Report #UCB/EECS-2017-90, University of California, Berkeley, Berkeley, CA, 2017, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-90.html.

[47] D. Potts, *Schnelle Fourier-Transformationen für nichtäquidistante Daten und Anwendungen*, Habilitationssrchift, University of Lübeck, Lübeck, Germany, 2003.

[48] D. Potts, G. Steidl, and M. Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, in Modern Sampling Theory: Mathematics and Applications, Birkhäuser Boston, Boston, 2001, pp. 247–270.

[49] W. H. Press and G. B. Rybicki, *Fast algorithm for spectral analysis of unevenly sampled data*, Astrophys. J., 338 (1989), pp. 277–280.

[50] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C++*, 3rd ed., Cambridge University Press, Cambridge, 2007.

[51] A. Prudnikov, Y. A. Brychkov, and O. I. Marichev, *Integrals and Series*, Volume 1. Elementary Functions, Gordon and Breach, New York, 1986.

[52] A. Prudnikov, Y. A. Brychkov, and O. I. Marichev, *Integrals and Series*, Volume 2. Special Functions, Gordon and Breach, New York, 1986.

[53] D. Ruis-Antolín and A. Townsend, *A nonuniform fast Fourier transform based on low rank approximation*, SIAM J. Sci. Comput., 40 (2018), pp. A529–A547, https://doi.org/10.1137/17M1134822.

[54] D. Slepian, *Some asymptotic expansions for prolate spheroidal wave functions*, J. Math. and Phys., 44 (1965), pp. 99–140, https://doi.org/10.1002/sapm196544199.

[55] D. Slepian, *Some comments on Fourier analysis, uncertainty, and modeling*, SIAM Rev., 25 (1983), pp. 379–393, https://doi.org/10.1137/1025078.

[56] G. Steidl, *A note on fast Fourier transforms for nonequispaced grids*, Adv. Comput. Math., 9 (1998), pp. 337–352.

[57] B. P. Sutton, D. C. Noll, and J. A. Fessler, *Fast, iterative image reconstruction for MRI in the presence of field inhomogeneities*, IEEE Trans. Med. Imag., 22 (2003), pp. 178–188.

[58] A. R. Thompson and R. N. Bracewell, *Interpolation and Fourier transformation of fringe visibilities*, Astronom. J., 79 (1974), pp. 11–24.

[59] M. Uecker and M. Lustig, *BART Toolbox for Computational Magnetic Resonance Imaging*, 2016, https://mrirecon.github.io/bart/.

[60] A. Viswanathan, A. Gelb, D. Cochran, and R. Renaut, *On reconstruction from nonuniform spectral data*, J. Sci. Comput., 45 (2010), pp. 487–513.

[61] T. Volkmer, *OpenMP Parallelization in the NFFT Software Library*, preprint 2012-07, Faculty of Mathematics, Technische Universität Chemnitz, Chemnitz, Germany, 2012.

[62] A. F. Ware, *Fast approximate Fourier transforms for irregularly spaced data*, SIAM Rev., 40 (1998), pp. 838–856, https://doi.org/10.1137/S003614459731533X.

[63] K. Zhang and J. U. Kang, *Graphics processing unit accelerated non-uniform fast Fourier transform for ultrahigh-speed, real-time Fourier-domain OCT*, Opt. Express, 18 (2010), pp. 23472–23487.

[64] Z. Zhao, Y. Shkolnisky, and A. Singer, *Fast steerable principal component analysis*, IEEE Trans. Comput. Imaging, 2 (2016), pp. 1–12.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.