SIAM J. SCI. COMPUT.
Vol. 35, No. 4, pp. C411–C437

PARALLEL THREE-DIMENSIONAL NONEQUISPACED FAST FOURIER TRANSFORMS AND THEIR APPLICATION TO PARTICLE SIMULATION*

MICHAEL PIPPIG† AND DANIEL POTTS†

Abstract. Starting from an approved serial algorithm, we develop a new parallel algorithm for calculating nonequispaced fast Fourier transforms on massively parallel distributed memory architectures. We demonstrate how to deal with the inherent load imbalance of the serial algorithm due to the use of oversampled FFT. This algorithm has been implemented in a new open source software library called PNFFT. Furthermore, we derive a new parallel distributed memory algorithm for the fast computation of fully Coulomb interactions in a charged particle system with nonperiodic boundary conditions based on a particle-mesh approximation scheme. We show that an appropriate adjustment of the underlying parallel nonequispaced fast Fourier transform circumvents severe load imbalance due to particle scaling. To prove the high scalability of our algorithms we provide performance results on a BlueGene/P system using up to 65536 cores.

Key words. parallel nonequispaced fast Fourier transform, parallel fast summation, parallel particle-mesh methods, NFFT

AMS subject classifications. 65T50, 65Y05

DOI. 10.1137/120888478

1. Introduction.

A broad variety of mathematical algorithms and applications depends on the calculation of the nonequispaced discrete Fourier transform (NDFT), which is a generalization of the discrete Fourier transform to nonequispaced nodes. Especially, its fast approximate realization called nonequispaced fast Fourier transform (NFFT) [8, 3, 55, 59, 52, 20, 31] led to the development of a large number of fast numerical algorithms, e.g., in computerized tomography [16, 9], particle simulation [50, 23], and spectral methods on adaptive grids, just to name a few examples. An extensive list of applications can be found e.g., in [20].

Roughly speaking, the NFFT consists of three steps. First, a deconvolution in frequency domain. Second, a fast Fourier transform (FFT) and, finally, a discrete convolution in spatial domains. The deconvolution and convolution is performed with a window function that is well localized in frequency and spatial domains. Therefore, these two convolution steps can be performed approximately in a fast way. Another advantage of the good localization is that parallel implementations of the convolution steps only require next neighbor communication.

The FFT plays a central role in the modular structure of the NFFT algorithm and is a perfect example for the important interplay between the development of fast algorithms and sustainable software engineering in order to produce high performance software. Without a doubt, the FFTW software library [18, 19] is an outstanding implementation of the FFT and one of the most important software packages in scientific computing. It offers support of shared memory parallelism and also distributed memory parallelism based on a one-dimensional decomposition of the input

*Submitted to the journal's Software and High-Performance computing section August 17, 2012; accepted for publication (in revised form) June 11, 2013; published electronically August 13, 2013. This work was partly supported by the German Ministery of Science and Education (BMBF) under grant 01IH08001B.

http://www.siam.org/journals/sisc/35-4/88847.html

†Chemnitz University of Technology, Department of Mathematics, 09107 Chemnitz, Germany (michael.pippig@mathematik.tu-chemnitz.de, potts@mathematik.tu-chemnitz.de).

array. However, it has been argued that the one-dimensional data decomposition is not scalable enough for modern, massively parallel, distributed memory architectures [6, 11, 14, 56, 1]. Whenever the dimensionality of the input array is at least three, a more scalable two-dimensional domain decomposition can be applied. Several publicly available, parallel FFT software libraries [48, 47, 41, 40, 36, 35, 45, 43] based on this approach have been proposed during the last years.

Following the example of the FFTW software library, the NFFT algorithm has been implemented in the publicly available NFFT software library [31, 30], which also offers support of shared memory parallelism [58] and a parallel implementation for graphic processing units [32]. However, to our knowledge there is no publicly available implementation of the NFFT algorithm suitable for distributed memory parallelism. The algorithms in this paper and their publicly available implementations are intended to close this gap between NFFT and modern distributed memory architectures.

In this paper we propose a parallel algorithm for computing the NFFT on massively parallel distributed memory architectures. This algorithm strongly requires the parallel pruned FFT [45] in order to overcome severe load balancing problems. Our highly scalable implementation is based on the message passing interface [39] and utilizes the PFFT software library [43] for computing the pruned FFT in parallel. Furthermore, we describe a massively parallel fast summation algorithm based on the parallel NFFT. The fast summation algorithm [50, 51] deals with the computation of Coulomb interactions in charged particle systems with nonperiodic boundary conditions. This is similar to Ewald-like particle-mesh algorithms, which only work for periodic boundary conditions; see, e.g., [21, Chap. 7] for an overview. Indeed, in [46] we point out that the building blocks of the fast summation for nonperiodic boundary conditions and the NFFT-based fast Ewald summation [23] for periodic boundary conditions are very similar. Especially, both algorithms employ the NFFT in order to achieve a fast algorithm. For periodic boundary conditions, there has been a broad variety of publications about parallel particle-mesh algorithms; e.g., see [57, 49, 38, 42, 37, 24, 53]. However, these implementations use simple cutoff schemes in order to deal with Coulomb interactions for nonperiodic boundary conditions and thereby totally ignore the long range nature of these interactions.

To our knowledge, this is the first paper that presents results on distributed memory parallelization of particle-mesh algorithms with nonperiodic boundary conditions, i.e., we present an approximation scheme that allows the accurate computation of the Coulomb interactions up to a prescribed error in parallel. A numerical comparison with parallel implementation of other methods, e.g., an implementation of the fast multipole method [29], will be published elsewhere.

The outline of this paper is as follows: We start in section 2 with the introduction of the notation and definitions that we will use in the remainder of the paper. Next, we give the definition of the NDFT and explain the basic principles of the NFFT in section 3. In section 4 we present our parallel NFFT algorithm, which we apply in section 5 to develop a parallel algorithm for computing the Coulomb potentials and fields of a charged particle system. Section 6 contains performance evaluations of our publicly available, parallel implementation. Finally, we conclude the paper in section 7.

2. Notation, definitions, and assumptions. In this section we introduce the notation, basic definitions, and assumptions that are used throughout the entire paper. Assume the multibandwidth $\boldsymbol{N} = (N_0, N_1, N_2)^\top \in 2\mathbb{N}^3$. We define the multi-index set of possible frequencies $\mathcal{I}_{\boldsymbol{N}} := \{-\frac{N_0}{2}, \dots, \frac{N_0}{2} - 1\} \times \{-\frac{N_1}{2}, \dots, \frac{N_1}{2} - 1\} \times$

$\{-\frac{N_2}{2}, \dots, \frac{N_2}{2}-1\}$, the total number of frequencies $|\mathcal{I}_{\boldsymbol{N}}| = N_0 \cdot N_1 \cdot N_2$, and the three-dimensional torus $\mathbb{T}^3 := \mathbb{R}^3/\mathbb{Z}^3 \sim [-\frac{1}{2}, \frac{1}{2})^3$. For $|\mathcal{I}_{\boldsymbol{N}}|$ complex numbers $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}$, the trigonometric polynomial $f\colon \mathbb{T}^3 \to \mathbb{C}$ is given by

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}}. \tag{2.1}$$

Hereby, $\boldsymbol{k}\boldsymbol{x} := k_0 x_0 + k_1 x_1 + k_2 x_2$ denotes the canonical scalar product. The fast evaluation of f at arbitrarily chosen nodes $\boldsymbol{x}_j \in \mathbb{T}^3$, $j = 1, \dots, M$, with $M \in \mathbb{N}$, i.e.,

$$f_j := f(\boldsymbol{x}_j) = \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j}, \quad j = 1, \dots, M, \tag{2.2}$$

is known as the three-dimensional NFFT. Equation (2.2) can be written as a matrix-vector product,

$$\boldsymbol{f} = \boldsymbol{A}\hat{\boldsymbol{f}},$$

with the vectors $\boldsymbol{f} := (f_j)_{j=1,\dots,M} \in \mathbb{C}^M$, $\hat{\boldsymbol{f}} := (\hat{f}_{\boldsymbol{k}})_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \in \mathbb{C}^{|\mathcal{I}_{\boldsymbol{N}}|}$, and the nonequispaced Fourier matrix $\boldsymbol{A} := (\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j})_{j=1,\dots,M;\, \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \in \mathbb{C}^{M \times |\mathcal{I}_{\boldsymbol{N}}|}$. For clarity of presentation we write the multi-index $\boldsymbol{k}$ in order to address elements of vectors and matrices at the linearized index $k_2 + N_2 k_1 + N_2 N_1 k_0$. In general, the matrix $\boldsymbol{A}$ is not square. Even for the square case, it is usually not orthogonal. Therefore, the definition of an inverse NFFT is not canonical, but can be realized by an iterative method; see, e.g., [33]. Instead, it is customary to define the adjoint transform by the matrix-vector product

$$\hat{\boldsymbol{h}} = \boldsymbol{A}^{\mathsf{H}} \boldsymbol{f},$$

that is equivalent to the sums

$$\hat{h}_{\boldsymbol{k}} = \sum_{j=1}^{M} f_j \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j}, \qquad \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}, \tag{2.3}$$

with the vector $\hat{\boldsymbol{h}} := (\hat{h}_{\boldsymbol{k}})_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}}$. In addition, we are interested in the fast calculation of the gradients

$$\nabla f_j := \nabla f(\boldsymbol{x}_j) = \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \nabla \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j}, \quad j = 1, \dots, M. \tag{2.4}$$

Equation (2.4) can be written as a matrix-vector product,

$$\nabla \boldsymbol{f} = \nabla \boldsymbol{A}\hat{\boldsymbol{f}},$$

with the vectors $\nabla \boldsymbol{f} := (\nabla f_j)_{j=1,\dots,M} \in \mathbb{C}^{3M}$, $\hat{\boldsymbol{f}} := (\hat{f}_{\boldsymbol{k}})_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \in \mathbb{C}^{|\mathcal{I}_{\boldsymbol{N}}|}$, and the matrix $\nabla \boldsymbol{A} := (\nabla \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}_j})_{j=1,\dots,M;\, \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \in \mathbb{C}^{3M \times |\mathcal{I}_{\boldsymbol{N}}|}$, where each gradient stands for three successive rows of the matrix.

The parallel NFFT (PNFFT) algorithms, implemented in our library [44], are fast approximate algorithms for computing the sums in (2.2) and the adjoint transform (2.3). Both these transforms are also the cornerstone for the nonequispaced convolution; see, e.g., [34]. In addition, we implemented a fast approximate algorithm for computing the gradients (2.4).

3. The three-dimensional NFFT algorithm. This section summarizes the mathematical theory and ideas behind the NFFT based on [52, 31]. For further NFFT approaches see [31, Appendix C]. Assume $\boldsymbol{n} = (n_0, n_1, n_2)^\top \in 2\mathbb{N}^3$, with $\boldsymbol{N} \leq \boldsymbol{n}$. Hereby, the inequality holds componentwise. Again, we define the multi-index set $\mathcal{I}_{\boldsymbol{n}} := \{-\frac{n_0}{2}, \dots, \frac{n_0}{2} - 1\} \times \{-\frac{n_1}{2}, \dots, \frac{n_1}{2} - 1\} \times \{-\frac{n_2}{2}, \dots, \frac{n_2}{2} - 1\}$ and the total number of frequencies $|\mathcal{I}_{\boldsymbol{n}}| = n_0 \cdot n_1 \cdot n_2$. Let $\psi\colon \mathbb{T} \to \mathbb{R}$ be a smooth window function, i.e., a function that is well localized in spatial domain and in frequency domain. We denote its Fourier coefficients by $\hat{\psi}_k$, $k \in \mathbb{Z}$. Furthermore, we define a multivariate window function $\varphi\colon \mathbb{T}^3 \to \mathbb{R}$ by the tensor product $\varphi(\boldsymbol{x}) := \psi(x_0)\psi(x_1)\psi(x_2)$. A simple consequence is that its Fourier coefficients

$$\hat{\varphi}_{\boldsymbol{k}} := \int_{\mathbb{T}^3} \varphi(\boldsymbol{x}) \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}} \mathrm{d}\boldsymbol{x}$$

are given by $\hat{\varphi}_{\boldsymbol{k}} = \hat{\psi}_{k_0}\hat{\psi}_{k_1}\hat{\psi}_{k_2}$, $\boldsymbol{k} = (k_0, k_1, k_2)^\top \in \mathbb{Z}^3$ with $\hat{\psi}_k := \int_{\mathbb{T}} \psi(x) \mathrm{e}^{+2\pi \mathrm{i} k x} \mathrm{d}x$, and the gradient $\nabla\varphi(\boldsymbol{x})$ can be easily computed by

$$\nabla\varphi(\boldsymbol{x}) = \left(\psi'(x_0)\psi(x_1)\psi(x_2),\, \psi(x_0)\psi'(x_1)\psi(x_2),\, \psi(x_0)\psi(x_1)\psi'(x_2)\right)^\top . \tag{3.1}$$

We follow the general approach of [55, 52] and approximate the complex exponentials in the trigonometric polynomial (2.1) by

$$\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}} \approx \frac{1}{|\mathcal{I}_{\boldsymbol{n}}| \hat{\varphi}_{\boldsymbol{k}}} \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x})} \varphi\left(\boldsymbol{x} - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{l} \odot \boldsymbol{n}^{-1}\right)},$$

where the multi-index set

$$\mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}) := \{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n}} \colon \boldsymbol{n} \odot \boldsymbol{x} - m\mathbf{1} \leq \boldsymbol{l} \leq \boldsymbol{n} \odot \boldsymbol{x} + m\mathbf{1}\}$$

collects these indexes where the window function φ is mostly concentrated. Here, $m \in \mathbb{N}$ is a small window cutoff parameter, which depends on the particular choice of the window function. We use the vector $\mathbf{1} := (1, 1, 1)^\top$, the componentwise vector product $\boldsymbol{n} \odot \boldsymbol{x} := (n_0 x_0, n_1 x_1, n_2 x_2)^\top$, the reciprocal of a vector $\boldsymbol{n}$ with nonzero components $\boldsymbol{n}^{-1} := (n_0^{-1}, n_1^{-1}, n_2^{-1})^\top$, and the inequality between two vectors holds componentwise.

After changing the order of summation in (2.1) we obtain for $\boldsymbol{x}_j \in \mathbb{T}^3$, $j = 1, \dots, M$, the approximation

$$f(\boldsymbol{x}_j) \approx \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} \left(\sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \frac{\hat{f}_{\boldsymbol{k}}}{|\mathcal{I}_{\boldsymbol{n}}| \hat{\varphi}_{\boldsymbol{k}}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{l} \odot \boldsymbol{n}^{-1}\right)} \right) \varphi\left(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right),$$

which causes a truncation error and an aliasing error; see [52, 31] for details. As can be readily seen, after an initial deconvolution step,

$$\hat{g}_{\boldsymbol{k}} = \frac{\hat{f}_{\boldsymbol{k}}}{|\mathcal{I}_{\boldsymbol{n}}| \hat{\varphi}_{\boldsymbol{k}}}, \quad \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}, \tag{3.2}$$

the expression in brackets can be computed via a three-dimensional oversampled FFT of total size $|\mathcal{I}_{\boldsymbol{n}}|$,

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{l} \odot \boldsymbol{n}^{-1}\right)}, \quad \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n}}. \tag{3.3}$$

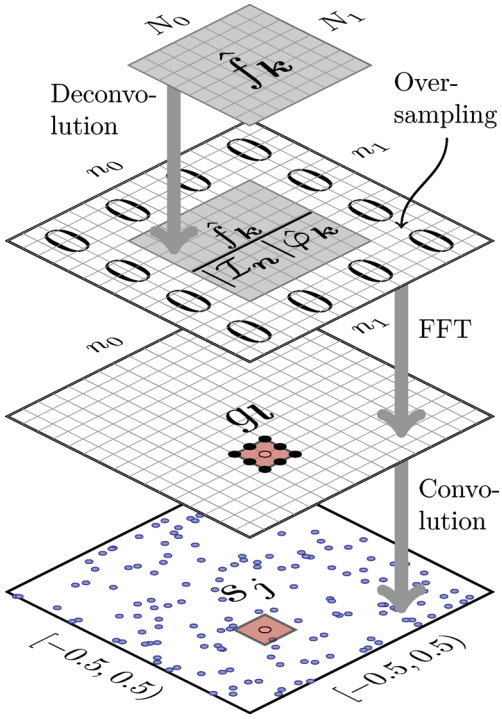

FIG. 3.1. *Two-dimensional illustration of the serial NFFT workflow for* $\boldsymbol{N} = (8,8)^\top$ *given Fourier coefficients, oversampled FFT size* $\boldsymbol{n} = (16,16)^\top$, $M = 150$ *nonequispaced nodes, and window cutoff parameter* $m = 1$. *At the beginning, the given Fourier coefficients* $\hat{f}_{\boldsymbol{k}}$ *are pointwise multiplied according to the deconvolution formula* (3.2) *and mapped into an oversampled FFT array of size* $\boldsymbol{n}$. *Afterwards, an FFT of size* $\boldsymbol{n}$ *is performed according to* (3.3). *The computation of the convolution sums* (3.4) *is illustrated at the example of a single red circled node* $\boldsymbol{x}_j$. *Note, that a small number of* $2m+1$ *(black circled) grid points is sufficient in order to compute the convolution sum* (3.4) *corresponding to this node. The support of the truncated window function centered at node* $\boldsymbol{x}_j$ *is given by the red rectangle of size* $(2m)^2$.

The final step consists of the evaluation of sums having at most $(2m+1)^3$ terms where the window function φ is sampled only in the neighborhood of the node $\boldsymbol{x}_j$, i.e.,

$$f(\boldsymbol{x}_j) \approx s_j := \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \varphi\left(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right) \tag{3.4}$$

and

$$\nabla f(\boldsymbol{x}_j) \approx \nabla s_j := \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \nabla\varphi\left(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right) .$$

In addition to the evaluation of the window function φ, it requires roughly $|\mathcal{I}_{\boldsymbol{N}}| + |\mathcal{I}_{\boldsymbol{n}}| \log |\mathcal{I}_{\boldsymbol{n}}| + (2m+1)^3 M$ floating point operations. We present a two-dimensional illustration of the serial NFFT algorithm for chosen parameters $\boldsymbol{N} = (8,8)^\top$, $\boldsymbol{n} = (16,16)^\top$, $M = 150$, and $m = 1$ in Figure 3.1. In matrix-vector notation, the NFFT algorithm can be written as $\boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{B}\boldsymbol{F}\boldsymbol{D}\hat{\boldsymbol{f}}$, where $\boldsymbol{D}$ is a real $|\mathcal{I}_{\boldsymbol{N}}| \times |\mathcal{I}_{\boldsymbol{N}}|$ diagonal matrix defined by

$$\boldsymbol{D} := \operatorname{diag}\left(1/\hat{\varphi}_{\boldsymbol{k}}\right)_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} .$$

The matrix $\boldsymbol{F} := (\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}(\boldsymbol{l} \odot \boldsymbol{n}^{-1})})_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n}}, \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}}$ is a truncated Fourier matrix of size $|\mathcal{I}_{\boldsymbol{n}}| \times |\mathcal{I}_{\boldsymbol{N}}|$ and $\boldsymbol{B}$ denotes the sparse real $M \times |\mathcal{I}_{\boldsymbol{n}}|$ matrix

$$\boldsymbol{B} := (b_{j\boldsymbol{l}})_{j=1,\ldots,M; \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n}}}, \quad b_{j\boldsymbol{l}} := \begin{cases} \varphi\left(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right) & : \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \\ 0 & : \boldsymbol{l} \notin \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \end{cases}.$$

An approximation of the adjoint transform is given by $\boldsymbol{A}^{\mathsf{H}} \boldsymbol{f} \approx \boldsymbol{D} \boldsymbol{F}^{\mathsf{H}} \boldsymbol{B}^{\top} \boldsymbol{f}$. The gradients (2.4) can be approximated by means of the analytic derivative of the window function [12], i.e., $\nabla \boldsymbol{A} \hat{\boldsymbol{f}} \approx \nabla \boldsymbol{B} \boldsymbol{F} \boldsymbol{D} \hat{\boldsymbol{f}}$ with

$$\nabla \boldsymbol{B} := (\nabla b_{j\boldsymbol{l}})_{j=1,\ldots,M; \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n}}}, \quad \nabla b_{j\boldsymbol{l}} := \begin{cases} \nabla \varphi\left(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}\right) & : \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \\ \boldsymbol{0} & : \boldsymbol{l} \notin \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \end{cases}.$$

Note that the NFFT and gradient NFFT only differ in the multiplication with the last matrix $\boldsymbol{B}$ and $\nabla \boldsymbol{B}$, respectively. Since the window function φ is defined as a tensor product, the evaluation of function values for both matrices can be easily combined. As we can see in (3.1) for a given node $\boldsymbol{x} = (x_0, x_1, x_2)^{\top} \in \mathbb{T}^3$ it is sufficient to evaluate the one-dimensional window function ψ and its derivative ψ' at the three coordinates x_0, x_1, x_2.

To keep the approximation error small, several functions φ with good localization in spatial and frequency domain have been proposed. In our parallel NFFT implementation the user is free to choose between the (dilated) *Gaussian* [8, 55, 7], (dilated) *cardinal central B–splines* [3, 55], and (dilated) *Kaiser–Bessel functions* [27, 17]. We point out that the approximation error introduced by the NFFT decays exponentially with the number of summands m. Error estimates for the multivariate case were presented in [10]; see also [31, Appendix C]. In the case of the Gaussian window function, the evaluations of the exponential function `exp()` can be reduced substantially by fast Gaussian gridding; see [20] and [31, Appendix C]. However, all proposed window functions can be evaluated with the same efficiency, if interpolation from short precomputed interpolation tables is used; see also section 6.1 for more details.

4. The parallel three-dimensional NFFT algorithm. In this section we describe a parallel algorithm for computing the three-dimensional NFFT on massively parallel, distributed memory architectures. The implementation of this algorithm is based on the massage passing interface [39]. Our parallel NFFT (PNFFT) algorithm combines the serial three-dimensional NFFT algorithm from section 3 with a three-dimensional block domain decomposition. In addition, we pay special attention to the case where all the nonequispaced nodes $\boldsymbol{x}_j$ are contained in a special subset of the torus $\mathbb{T}^3$. For a given node scaling factor $\boldsymbol{C} = (C_0, C_1, C_2)^{\top} \in (0,1]^3$ we define the truncated torus $\mathbb{T}^3_{\boldsymbol{C}} := [-\frac{C_0}{2}, \frac{C_0}{2}) \times [-\frac{C_1}{2}, \frac{C_1}{2}) \times [-\frac{C_2}{2}, \frac{C_2}{2})$. For the PNFFT we assume $\boldsymbol{x}_j \in \mathbb{T}^3_{\boldsymbol{C}}$ for every $j = 1, \ldots, M$. Obviously, for $C_0 = C_1 = C_2 = 1$ this corresponds to the serial NFFT, where the nodes $\boldsymbol{x}_j$ are contained in the whole three-dimensional torus $\mathbb{T}^3$. This slight generalization is necessary in order to assure a load balanced distribution of nodes $\boldsymbol{x}_j$ whenever the nodes are concentrated in the center of the box as we will see in the next section.

4.1. Description of the algorithm. Assume $\boldsymbol{P} = (P_0, P_1, P_2)^{\top} \in \mathbb{N}^3$. We identify every MPI process of a given parallel hardware architecture with a multiindex of the three-dimensional process mesh $\mathcal{P}_{\boldsymbol{P}} := \{0, \ldots, P_0 - 1\} \times \{0, \ldots, P_1 - 1\} \times$

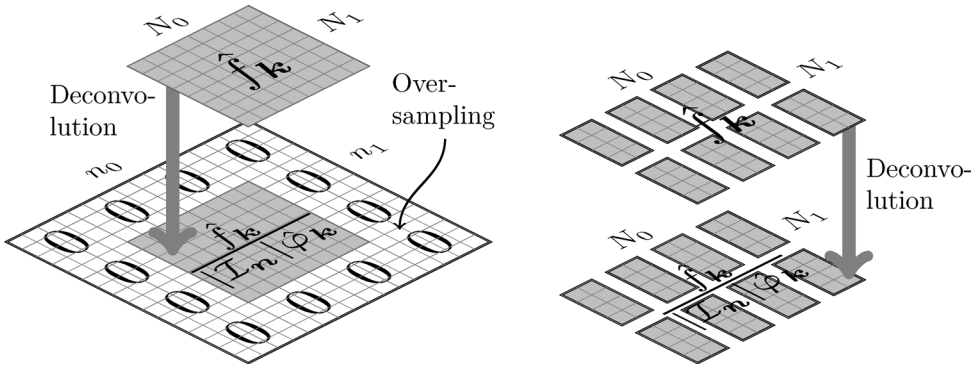

FIG. 4.1. *Two-dimensional illustration of the serial deconvolution workflow according to* (3.2) *on the left and the parallel deconvolution workflow according to* (4.1) *on the right. We chose* $\boldsymbol{N} = (8,8)^\top$ *given Fourier coefficients, oversampled FFT size* $\boldsymbol{n} = (16,16)^\top$*, and a process mesh of size* $\boldsymbol{P} = (4,2)^\top$*. The serial algorithm uses explicit mapping of the incoming Fourier coefficients* $\hat{f}_{\boldsymbol{k}}$ *into an oversampled FFT array of size* $\boldsymbol{n}$ *filled with zeros. In contrast, our parallel data distribution avoids the distribution of zeros and therefore does not depend on the oversampled FFT size* $\boldsymbol{n}$*.*

$\{0,\dots,P_2-1\}$. For every process $\boldsymbol{r} = (r_0,r_1,r_2)^\top \in \mathcal{P}_{\boldsymbol{P}}$ we define the multi-index set

$$\mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}} = \left\{ (k_0,k_1,k_2)^\top \in \mathcal{I}_{\boldsymbol{N}} : -\frac{N_t}{2} + r_t\frac{N_t}{P_t} \le k_t < -\frac{N_t}{2} + (r_t+1)\frac{N_t}{P_t},\ t = 0,1,2 \right\}.$$

At the beginning of our parallel algorithm, we assume the NFFT input array of $|\mathcal{I}_{\boldsymbol{N}}|$ complex numbers to be distributed among the three-dimensional process mesh $\mathcal{P}_{\boldsymbol{P}}$ such that every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ holds the input data $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$, in its locally available memory. For the sake of simplicity, we assume that the input array sizes N_0, N_1, N_2 are divisible by the process mesh sizes P_0, P_1, P_2. Therefore, the input array is distributed in equal blocks of size $N_0/P_0 \cdot N_1/P_1 \cdot N_2/P_2$. This restriction serves to keep the notation simple. Nevertheless, our implementation supports arbitrary input array sizes $\boldsymbol{N} \in \mathbb{N}^3$ and process mesh sizes $\boldsymbol{P} \in \mathbb{N}^3$ regardless of their divisibility.

The serial NFFT algorithm starts with the deconvolution step (3.2) that consists of an ordinary pointwise multiplication. It can be calculated straightforwardly in parallel, i.e., every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ computes

$$\hat{g}_{\boldsymbol{k}} = \frac{\hat{f}_{\boldsymbol{k}}}{|\mathcal{I}_{\boldsymbol{n}}|\hat{\varphi}_{\boldsymbol{k}}}, \quad \boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}\,. \tag{4.1}$$

Figure 4.1 shows a two-dimensional illustration of the serial deconvolution workflow according to (3.2) and the parallel deconvolution workflow according to (4.1) for parameters $\boldsymbol{N} = (8,8)^\top$, $\boldsymbol{n} = (16,16)$, and a process mesh of size $\boldsymbol{P} = (4,2)^\top$.

In the second step (3.3) we compute a three-dimensional oversampled FFT. Before we look at the parallel counter part of this step, we need the following slight generalization. Similar to the truncated input data set of an oversampled FFT, we want to allow a truncated output data set. Therefore, we introduce the pruned FFT output size $\boldsymbol{L} \in 2\mathbb{N}^3$ with $\boldsymbol{L} \le \boldsymbol{n}$. Hereby, the inequality holds componentwise. The pruned FFT is given by

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{l} \odot \boldsymbol{n}^{-1}\right)}, \quad \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{L}}\,,$$

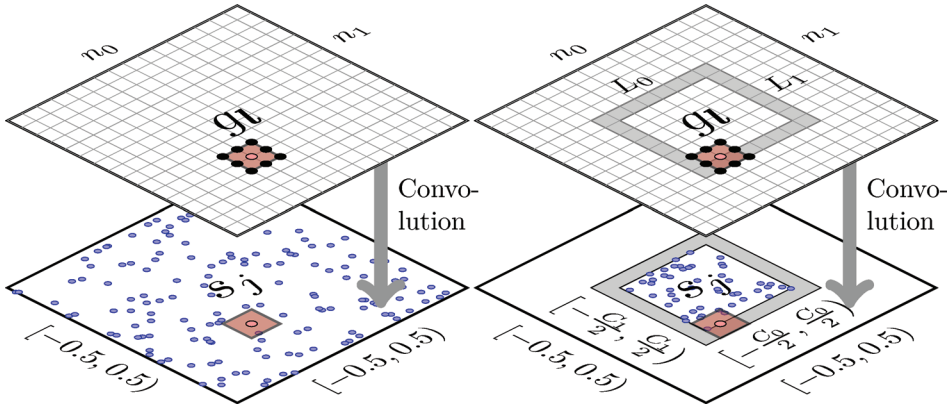

FIG. 4.2. *Two-dimensional illustration of the serial convolution workflow according to* (3.4) *for node scaling factor* $\boldsymbol{C} = (1,1)^\top$ *and* $M = 150$ *nonequispaced nodes on the left and node scaling factor* $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$ *and* $M = 50$ *nonequispaced nodes on the right. Furthermore, we chose an oversampled FFT size* $\boldsymbol{n} = (16,16)^\top$*, pruned FFT output size* $\boldsymbol{L} = (8,8)^\top$*, and window cutoff parameter* $m = 1$*. In both cases the computation of the convolution sums* (3.4) *is illustrated at the example of a single red circled node* $\boldsymbol{x}_j$*. The support of the truncated window function centered at node* $\boldsymbol{x}_j$ *is given by the red rectangle of size* $(2m)^2$*. Again, a small number of* $2m+1$ *(black circled) grid points is sufficient in order to compute the convolution sum* (3.4) *corresponding to this node. However, on the left for every datum* $g_{\boldsymbol{l}}$ *there exists at least one node* $\boldsymbol{x}_j$ *that depends on* $g_{\boldsymbol{l}}$ *for the computation of the corresponding convolution sum* s_j*. In contrast, on the right the nodes have been scaled to the center of the unit box. Therefore, no data* $g_{\boldsymbol{l}}$ *outside the gray colored border are ever needed for the computation of any convolution sum* s_j*. The gray border of width* m *results from the overlapping support of the window function.*

with $\mathcal{I}_{\boldsymbol{L}} := \{-\frac{L_0}{2}, \dots, \frac{L_0}{2}-1\} \times \{-\frac{L_1}{2}, \dots, \frac{L_1}{2}-1\} \times \{-\frac{L_2}{2}, \dots, \frac{L_2}{2}-1\}$. Obviously, for $\boldsymbol{L} = \boldsymbol{n}$ the pruned FFT coincides with the second step of the serial NFFT algorithm given by (3.3). The significance of the pruned FFT output size $\boldsymbol{L}$ becomes clear, if we have a look at the third step of the NFFT algorithm shown in (3.4). There the summation runs over the multi-index sets $\mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \subset \mathcal{I}_{\boldsymbol{n}}$, $j = 1, \dots, M$. We want to take advantage of the fact that all the nodes $\boldsymbol{x}_j$ are contained in the truncated torus $\mathbb{T}^3_{\boldsymbol{C}}$. In order to avoid the computation of unneeded coefficients $g_{\boldsymbol{l}}$ we are looking for the smallest multi-index set $\mathcal{I}_{\boldsymbol{L}}$ that holds $\mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \subset \mathcal{I}_{\boldsymbol{L}}$ for every $j = 1, \dots, M$. The componentwise smallest $\boldsymbol{L} = (L_0, L_1, L_2)^\top \in 2\mathbb{N}^3$ that fulfills these conditions is given by

$$L_t := \min\left\{n_t, 2\left(\left\lceil C_t \frac{n_t}{2}\right\rceil + m\right)\right\}, \quad t = 0,1,2\,.$$

Figure 4.2 shows a two-dimensional illustration of the additional condition $\boldsymbol{x}_j \in \mathbb{T}^3_{\boldsymbol{C}}$ and its implications on the serial computation of the convolution sums (3.4) for parameters $\boldsymbol{n} = (16,16)^\top$, $\boldsymbol{L} = (8,8)^\top$, $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$, and $m = 1$.

In parallel we substitute the three-dimensional pruned FFT by a parallel one, i.e.,

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{g}_{\boldsymbol{k}}\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}(\boldsymbol{l} \odot \boldsymbol{n}^{-1})}, \quad \boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}}\,. \tag{4.2}$$

The formal order of summation in this notation was chosen to symbolize the parallel data decomposition of a block distributed parallel three-dimensional FFT algorithm. In general, a parallel FFT algorithm may use a different order of summation or an

approximate algorithm to calculate the Fourier transform. The inner sum reflects that every process $\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}$ starts with calculations on its locally available input data block of size $N_0/P_0 \cdot N_1/P_1 \cdot N_2/P_2$. The outer sum stands for the global communication that must be performed somehow within the parallel FFT algorithm. After the parallel FFT the output data $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{L}}$, is distributed on the process mesh in a similar way as the input data set, i.e., every process owns a block of $L_0/P_0 \cdot L_1/P_1 \cdot L_2/P_2$ complex numbers. Once more, we assume that L_0, L_1, L_2 are divisible by the process mesh sizes P_0, P_1, P_2 in order to keep the notation simple. Again, for every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ the multi-index set

$$\mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}} := \left\{ (k_0, k_1, k_2)^\top \in \mathcal{I}_{\boldsymbol{n}} : -\frac{L_t}{2} + r_t \frac{L_t}{P_t} \leq k_t < -\frac{L_t}{2} + (r_t + 1)\frac{L_t}{P_t},\ t = 0, 1, 2 \right\}$$

collects all the multi-indexes of locally available data. We apply the PFFT software library [43] for computing the parallel pruned FFT. This library was developed for calculating parallel FFT on massively parallel architectures. It uses a transpose FFT algorithm that consist of successive one-dimensional FFTs and global data transpositions; see [45] for details. We stress that PFFT is the only publicly available parallel FFT software library that pays special attention to the efficient parallel computation of pruned FFT. This feature is crucial in order to assure a good load balancing of our parallel NFFT algorithm. It is noteworthy to say that PFFT is based on a two-dimensional domain decomposition, i.e., the three-dimensional decomposed FFT input $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$, and output $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}}$ is redistributed before and after every parallel FFT. Therefore, an upper limit for the number of processes is given by the two-dimensional decomposition of the parallel FFT. Figure 4.3 shows a two-dimensional illustration of the serial FFT workflow according to (3.3) and the parallel pruned FFT workflow according to (4.2) for parameters $\boldsymbol{N} = (8, 8)^\top$, $\boldsymbol{n} = (16, 16)^\top$, and a process mesh of size $\boldsymbol{P} = (4, 2)^\top$. Note, that the parallel ghost cell duplication step (4.3) within this illustration will be explained in the next paragraphs.

The block data distribution of the FFT output $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{L}}$, naturally induces a block decomposition of the truncated Torus $\mathbb{T}^3_{\boldsymbol{C}}$. This motivates the definition of the index sets

$$\mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}} := \left\{ j = 1, \ldots, M : \exists \boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}} \text{ with } \boldsymbol{l} \leq \boldsymbol{n} \odot \boldsymbol{x}_j < \boldsymbol{l} + \boldsymbol{1} \right\}$$

for every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$. We assign all nodes $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$, to a single process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$. As one can already see, heterogeneous distributions of the nodes $\boldsymbol{x}_j$, $j = 1, \ldots, M$, can lead to imbalances in memory consumption and workload between the processes, which is a typical problem of mesh based domain decompositions.

According to the discrete convolution step of the serial NFFT algorithm, we compute the sums (3.4), which run over the local multi-index sets $\mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$. Our choice of parameter $\boldsymbol{L}$ assures $\mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j) \subset \mathcal{I}_{\boldsymbol{L}}$ for every $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$, i.e., the output of the pruned FFT is sufficient. But in general, not all sufficient data $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)$, are located on a single process $\boldsymbol{r}$. Therefore, we perform a communication step in order to gather all the additionally needed data. However, this step equals a mesh ghost cell communication [22, Chap. 5.6.1] and only involves nearest neighbor communication. With the definition of the multi-index sets

$$\mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P},m} := \Big\{ (l_0, l_1, l_2) \in \mathcal{I}_{\boldsymbol{n}} : \\ -\frac{L_t}{2} + r_t \frac{L_t}{P_t} - m \leq l_t < -\frac{L_t}{2} + (r_t + 1)\frac{L_t}{P_t} + m,\ t = 0, 1, 2 \Big\},$$

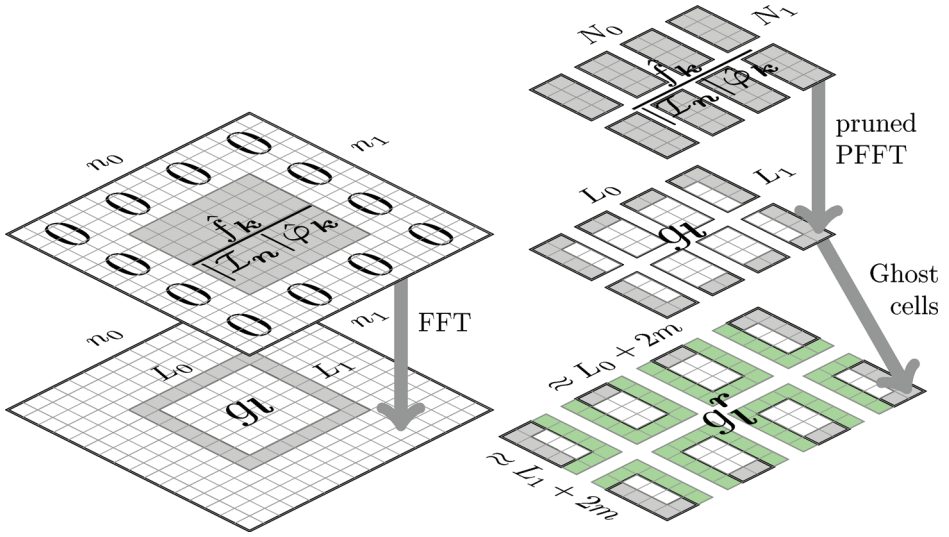

FIG. 4.3. *Two-dimensional illustration of the serial FFT workflow according to* (3.3) *on the left and the parallel pruned FFT workflow according to* (4.2) *on the right. We chose* $\boldsymbol{N} = (8,8)^\top$ *given Fourier coefficients, oversampled FFT size* $\boldsymbol{n} = (16,16)^\top$*, pruned FFT output size* $\boldsymbol{L} = (8,8)^\top$*, window cutoff parameter* $m = 1$*, and a process mesh of size* $\boldsymbol{P} = (4,2)^\top$*. A naive block decomposition of the FFT on the left would lead to several processes that own input blocks full of zeros before the FFT and output blocks full with unnecessary data* $g_{\boldsymbol{l}}$ *outside the gray colored border after the FFT. Instead, our algorithm uses a parallel pruned FFT that works with a block distribution of the necessary input and output data. Note, that the parallel NFFT needs to communicate a border of* m *(green colored) ghost cells according to* (4.3) *in order to prepare the parallel convolution according to* (4.4)*.*

for all processes $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$, we symbolize the ghost cell communication by

$$g^{\boldsymbol{r}}_{\boldsymbol{l}} = g_{\boldsymbol{l}}, \quad \boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P},m}\,. \tag{4.3}$$

Note that every process must gather the data $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P},m} \setminus \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}}$, from their nearest neighbors by explicit communication. We use the ghost cell support of the PFFT software library for implementing the ghost cell communication. Figure 4.3 also includes a two-dimensional illustration of the parallel ghost cell communication according to (4.3) with parameters $L = (8,8)^\top$, $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$, $m = 1$, and a process mesh of size $\boldsymbol{P} = (4,2)^\top$.

Finally, the sums

$$s_j = \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g^{\boldsymbol{r}}_{\boldsymbol{l}}\, \varphi(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1}), \quad j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}\,, \tag{4.4}$$

are calculated locally on all processes $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$. Figure 4.4 shows a two-dimensional illustration of the serial convolution workflow according to (3.4) and the parallel convolution workflow according to (4.4) for parameters $\boldsymbol{n} = (16,16)^\top$, $\boldsymbol{L} = (8,8)^\top$, $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$, $M = 50$, $m = 1$, and a process mesh of size $\boldsymbol{P} = (4,2)^\top$. Algorithm 1 summarizes the PNFFT algorithm in pseudocode and Figure 4.5 gives a summarizing two-dimensional illustration of the serial and parallel NFFT workflow for parameters $\boldsymbol{N} = (8,8)^\top$, $\boldsymbol{n} = (16,16)^\top$, $\boldsymbol{L} = (8,8)^\top$, $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$, $M = 50$, $m = 1$, and a process mesh of size $\boldsymbol{P} = (4,2)^\top$.

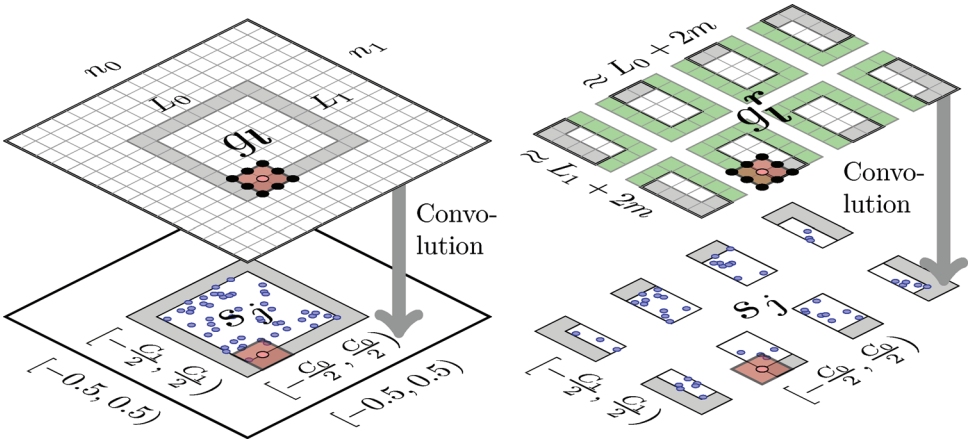

FIG. 4.4. *Two-dimensional illustration of the serial convolution workflow according to* (3.4) *on the left and the parallel convolution workflow according to* (4.4) *on the right. We chose an oversampled FFT size* $\boldsymbol{n} = (16, 16)^\top$*, pruned FFT output size* $\boldsymbol{L} = (8, 8)^\top$*, node scaling factor* $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$*,* $M = 50$ *nonequispaced nodes, window cutoff parameter* $m = 1$*, and a process mesh of size* $\boldsymbol{P} = (4, 2)^\top$*. In both cases the computation of the convolution sums* (3.4) *is illustrated at the example of a single red circled node* $\boldsymbol{x}_j$*. The support of the truncated window function centered at node* $\boldsymbol{x}_j$ *is given by the red rectangle of size* $(2m)^2$*. Again, a small number of* $2m + 1$ *(black circled) grid points is sufficient in order to compute the convolution sum* (3.4) *corresponding to this node. In the parallel case, the nonequispaced nodes have been block distributed according to the block distribution of the FFT outputs* $g_{\boldsymbol{l}}$*. Every process is able to compute the convolution sums* s_j *for all locally available nodes* $\boldsymbol{x}_j$ *from the locally available data* $g^{\boldsymbol{r}}_{\boldsymbol{l}}$*.*

ALGORITHM 1. PARALLEL, THREE-DIMENSIONAL, NFFT (PNFFT) FOR EACH PROCESS $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$.

Input: $\boldsymbol{x}_j \in \mathbb{T}^3_{\boldsymbol{C}}$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$, and $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$.

1: For $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$ compute $\hat{g}_{\boldsymbol{k}} := |\mathcal{I}_{\boldsymbol{n}}|^{-1} \cdot \hat{f}_{\boldsymbol{k}} / \hat{\varphi}_{\boldsymbol{k}}$.

2: For $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}}$ compute $g_{\boldsymbol{l}} := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} (\boldsymbol{l} \odot \boldsymbol{n}^{-1})}$ by a parallel, three-dimensional, pruned FFT.

3: For $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P},m}$ gather $g^{\boldsymbol{r}}_{\boldsymbol{l}} := g_{\boldsymbol{l}}$ by a ghost cell communication.

4: For $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$ compute $s_j := \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g^{\boldsymbol{r}}_{\boldsymbol{l}} \, \varphi(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1})$.

5: For $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$ compute $\nabla s_j := \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g^{\boldsymbol{r}}_{\boldsymbol{l}} \, \nabla \varphi(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1})$.

Output: Approximate function values $s_j \approx f_j$ and gradients $\nabla s_j \approx \nabla f_j$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$.

The adjoint PNFFT algorithm can be derived analogously from the serial adjoint NFFT algorithm [31]. Note that the adjoint counterpart of the ghost cell communication (4.3) is a sum over all ghost cells, i.e.,

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} g^{\boldsymbol{s}}_{\boldsymbol{l}}, \quad \boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}} \, .$$

The pseudocode of the adjoint PNFFT is given by Algorithm 2.

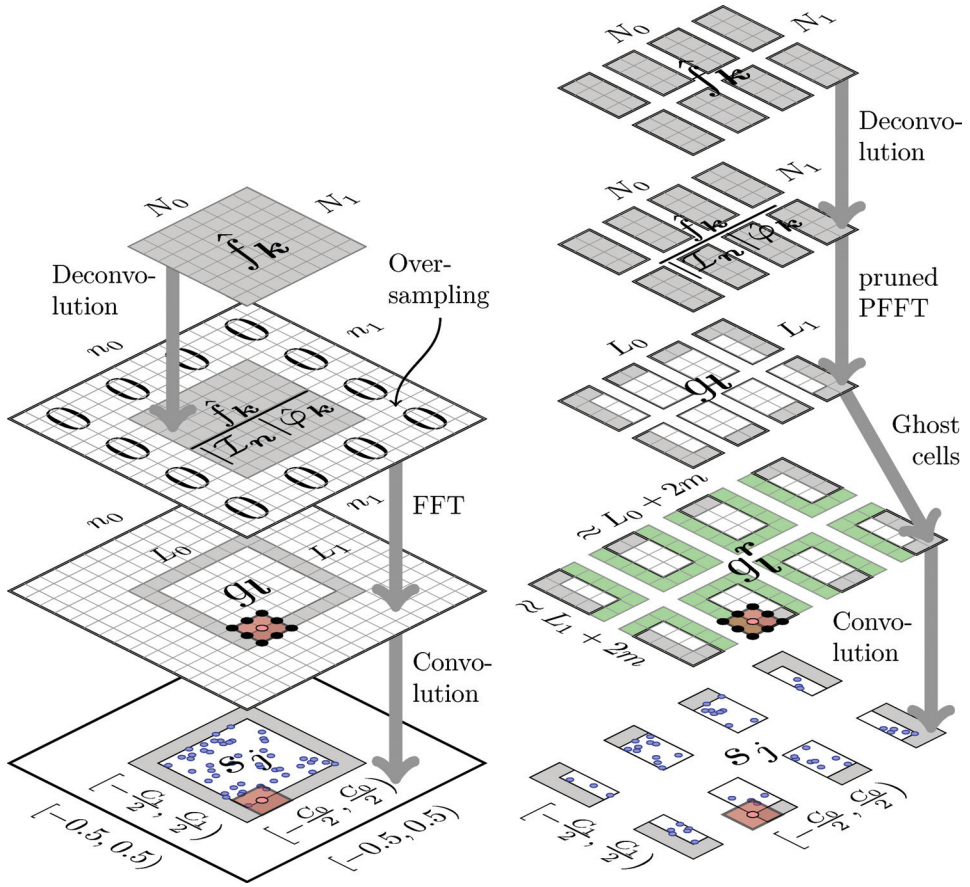

FIG. 4.5. *Two-dimensional illustration of the serial NFFT workflow on the left and the parallel NFFT workflow of Algorithm* 1 *on the right. We chose* $\boldsymbol{N} = (8,8)^\top$ *given Fourier coefficients, oversampled FFT size* $\boldsymbol{n} = (16,16)^\top$*, pruned FFT output size* $\boldsymbol{L} = (8,8)^\top$*, node scaling factor* $\boldsymbol{C} = (\frac{3}{8}, \frac{3}{8})^\top$*,* $M = 50$ *nonequispaced nodes, window cutoff parameter* $m = 1$*, and a process mesh of size* $\boldsymbol{P} = (4,2)^\top$*. For detailed explanations of the separate steps, see Figures* 4.1, 4.3*, and* 4.4.

5. Application of the parallel NFFT. In this section we demonstrate the application of our PNFFT algorithm in order to calculate the Coulomb potentials and forces of a charged particle system on massively parallel distributed memory architectures. We start with the outline of the serial fast summation algorithm [50, 51] and continue with its parallel counterpart.

5.1. Serial fast summation algorithm. Assume M charged particles with charge $q_j \in \mathbb{R}$ at position $\boldsymbol{x}_j \in \mathbb{T}^3$, $j = 1\ldots, M$. We are interested in the fast evaluation of the potentials

$$\phi_j := \phi(\boldsymbol{x}_j) = \sum_{\substack{l=1 \\ l\neq j}}^{M} q_l \frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2}, \quad j = 1, \ldots, M, \tag{5.1}$$

and fields

$$\boldsymbol{E}_j := -\nabla\phi(\boldsymbol{x}_j) = \sum_{\substack{l=1 \\ l\neq j}}^{M} q_l \frac{\boldsymbol{x}_j - \boldsymbol{x}_l}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2^3}, \quad j = 1, \ldots, M\,. \tag{5.2}$$

ALGORITHM 2. ADJOINT, PARALLEL, THREE-DIMENSIONAL, NFFT (ADJOINT PNFFT) FOR EACH PROCESS $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$.

Input: $\boldsymbol{x}_j \in \mathbb{T}^3_{\boldsymbol{C}}$, and $f_j \in \mathbb{C}$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$.

1: For $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P},m}$ compute $g^{\boldsymbol{r}}_{\boldsymbol{l}} := \sum_{\substack{\{j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}: \\ \boldsymbol{l} \in \mathcal{I}_{\boldsymbol{n},m}(\boldsymbol{x}_j)\}}} f_j \, \varphi(\boldsymbol{x}_j - \boldsymbol{l} \odot \boldsymbol{n}^{-1})$.

2: For $\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{L},\boldsymbol{P}}$ accumulate $g_{\boldsymbol{l}} := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} g^{\boldsymbol{s}}_{\boldsymbol{l}}$ by an adjoint ghost cell communication.

3: For $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$ compute $\hat{g}_{\boldsymbol{k}} := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{L},\boldsymbol{P}}} g_{\boldsymbol{l}} \, \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k}(\boldsymbol{l} \odot \boldsymbol{n}^{-1})}$ by an adjoint, parallel, three-dimensional, pruned FFT.

4: For $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$ compute $\hat{s}_{\boldsymbol{k}} := |\mathcal{I}_{\boldsymbol{n}}|^{-1} \cdot \hat{g}_{\boldsymbol{k}} / \hat{\varphi}_{\boldsymbol{k}}$.

Output: Approximate coefficients $\hat{s}_{\boldsymbol{k}} \approx \hat{h}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$.

Hereby, $\|\boldsymbol{x}\|_2 := (x_0^2 + x_1^2 + x_2^2)^{1/2}$ denotes the Euclidean norm. Without loss of generality we may assume that the nodes are scaled, such that $\|\boldsymbol{x}_j\|_2 < \frac{1}{4} - \frac{\varepsilon_{\mathrm{B}}}{2}$ with a small constant $\varepsilon_B > 0$ and consequently $\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2 < \frac{1}{2} - \varepsilon_{\mathrm{B}}$.

We now outline the NFFT-based fast summation algorithm for the fast computation of (5.1) and (5.2). It requires $\mathcal{O}(M \log M)$ arithmetic operations for uniformly distributed source nodes $\boldsymbol{x}_j$. This approach was suggested in [50, 51]. There, a regularization

$$R(r) := \begin{cases} T_{\mathrm{I}}(r) & \text{if} \quad r \le \varepsilon_{\mathrm{I}}, \\ T_{\mathrm{B}}(r) & \text{if} \quad \frac{1}{2} - \varepsilon_{\mathrm{B}} < r < \frac{1}{2}, \\ T_{\mathrm{B}}(\frac{1}{2}) & \text{if} \quad \frac{1}{2} \le r, \\ \frac{1}{r} & \text{otherwise}, \end{cases}$$

with a near field cutoff radius $\varepsilon_I > 0$ and a far field regularization border $\varepsilon_B > 0$ has been introduced. The functions T_{I} and T_{B} are chosen such that $R(\|\boldsymbol{x}\|_2)$ is in the Sobolev space $H^p(\mathbb{T}^3)$ for an appropriate degree of smoothness $p \in \mathbb{N}$. Several regularizations of $\frac{1}{r}$ are possible, e.g., by algebraic polynomials, splines, trigonometric polynomials, or two point Taylor interpolation; see [15]. The potentials ϕ_j in (5.1) can be approximated by

$$\phi_j \approx h_j := \phi_j^{\mathrm{NE}} + \phi_j^{\mathrm{RF}},$$

where

$$\phi_j^{\mathrm{NE}} := R(0) + \sum_{l \in \mathcal{I}^{\mathrm{NE}}_{\varepsilon_{\mathrm{I}}}(j)} q_l \left(\frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2} - R(\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2) \right), \tag{5.3}$$

$$\phi_j^{\mathrm{RF}} := \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{R}_{\boldsymbol{k}} \left(\sum_{l=1}^{M} q_l \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_l} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}. \tag{5.4}$$

Hereby, the index set $\mathcal{I}^{\mathrm{NE}}_{\varepsilon_{\mathrm{I}}}(j) := \{l \in \{1, \ldots, M\} \setminus \{j\} \colon \|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2 < \varepsilon_{\mathrm{I}}\}$ collects all the indexes of $\boldsymbol{x}_j$ next neighbors and the Fourier coefficients of the regularized kernel function

$$\hat{R}_{\boldsymbol{k}} := \frac{1}{|\mathcal{I}_{\boldsymbol{N}}|} \sum_{\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{N}}} R(\|\boldsymbol{l} \odot \boldsymbol{N}^{-1}\|_2) \mathrm{e}^{+2\pi \mathrm{i} (\boldsymbol{l} \odot \boldsymbol{N}^{-1}) \boldsymbol{k}}, \quad \boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}},$$

are precomputed by a three-dimensional adjoint discrete Fourier transform. Since the gradient of the regularization is given by $\nabla R(\|\boldsymbol{x}\|_2) = R'(\|\boldsymbol{x}\|_2)\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$, we are able to approximate the fields $\boldsymbol{E}_j$ in (5.2) analogously to the potentials, by

$$\boldsymbol{E}_j \approx -\nabla h_j := \boldsymbol{E}_j^{\mathrm{NE}} + \boldsymbol{E}_j^{\mathrm{RF}},$$

where

$$\boldsymbol{E}_j^{\mathrm{NE}} := -\nabla \phi_j^{\mathrm{NE}} = \sum_{l \in \mathcal{I}_{\varepsilon_{\mathrm{I}}}^{\mathrm{NE}}(j)} q_l \frac{\boldsymbol{x}_j - \boldsymbol{x}_l}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2} \left(\frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2^2} + R'(\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2) \right), \tag{5.5}$$

$$\boldsymbol{E}_j^{\mathrm{RF}} := -\nabla \phi_j^{\mathrm{RF}} = -\sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}} \hat{R}_{\boldsymbol{k}} \left(\sum_{l=1}^{M} q_l \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_l} \right) \nabla \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}. \tag{5.6}$$

If the nodes $\boldsymbol{x}_j$ are "sufficiently uniformly distributed" this can indeed be done in a fast way as shown below.

Near field computation (5.3), (5.5). To achieve the desired complexity of our algorithm we suppose that there exists a small constant $\nu \in \mathbb{N}$ such that the near field index sets $\mathcal{I}_{\varepsilon_{\mathrm{I}}}^{\mathrm{NE}}(j)$ contain at most ν indexes for every node $\boldsymbol{x}_j$, $j = 1, \ldots, M$. This implies that ε_{I} depends linearly on $1/\sqrt[3]{M}$. Then, for fixed $\boldsymbol{x}_j$ the sum (5.3) contains not more than ν terms so that its evaluation at all M nodes $\boldsymbol{x}_j$ requires only $\mathcal{O}(\nu M)$ arithmetic operations.

Far field computation (5.4), (5.6) by NFFT. The expression in the inner brackets of (5.4) can be computed by an adjoint, parallel, three-dimensional NFFT with total number of frequencies $|\mathcal{I}_{\boldsymbol{N}}|$. This is followed by $|\mathcal{I}_{\boldsymbol{N}}|$ multiplications with the Fourier coefficients $\hat{R}_{\boldsymbol{k}}$ of the regularized kernel function and completed by a parallel, three-dimensional NFFT with total number of frequencies $|\mathcal{I}_{\boldsymbol{N}}$— to compute the outermost sum. If m is the window cutoff parameter and $\boldsymbol{n}$ the FFT size of the (adjoint) NFFT, then the proposed evaluation of ϕ_j^{RF} and $\boldsymbol{E}_j^{\mathrm{RF}}$ at the nodes $\boldsymbol{x}_j$, $j = 1, \ldots, M$, requires $\mathcal{O}(m^3 M + |\mathcal{I}_{\boldsymbol{n}}| \log |\mathcal{I}_{\boldsymbol{n}}|))$ arithmetic operations. The relation between $M, \boldsymbol{N}$, and $\boldsymbol{n}$ is determined by the approximation error of the algorithm and is discussed in detail in [50, 51, 15].

5.2. Parallel fast summation algorithm. After we have seen the highly modularized structure of the serial fast summation algorithm it is easy to derive a parallel fast summation algorithm by substituting every module with its parallel counterpart. Our parallel algorithm starts with a parallel forward sorting step that assures the following two conditions. First, we need to distribute the nodes $\boldsymbol{x}_j$ according to our block decomposition such that every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ holds the nodes $\boldsymbol{x}_j$, $j \in \mathcal{M}_{\boldsymbol{P}}^{\boldsymbol{r}}$. Furthermore, every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ needs local copies of all the nodes that are involved in the calculation of the near field sums (5.3) and (5.5), i.e., all the nodes $\boldsymbol{x}_j$, $j \in \bigcup_{l \in \mathcal{M}_{\boldsymbol{P}}^{\boldsymbol{r}}} \mathcal{I}_{\varepsilon_{\mathrm{I}}}^{\mathrm{NE}}(l)$. We perform these two tasks at once using a fine-grained data distribution operation [26] that is implemented within a software library for parallel sorting algorithms [4].

We assume that all nodes $\boldsymbol{x}_j$ are located in a cubic box. Note that the serial fast summation algorithm requires the condition $\|\boldsymbol{x}_j\|_2 < \frac{1}{4} - \frac{\varepsilon_B}{2}$ for every node $\boldsymbol{x}_j$, $j = 1, \ldots, M$. A cubic box can have at most box length $\frac{1}{\sqrt{3}}(\frac{1}{2} - \varepsilon_B)$ in order to fit into a ball with radius $\frac{1}{4} - \frac{\varepsilon_B}{2}$. Therefore, we set the node scaling factor

$\boldsymbol{C} = \frac{1}{\sqrt{3}}(\frac{1}{4} - \frac{\varepsilon_B}{2}, \frac{1}{4} - \frac{\varepsilon_B}{2}, \frac{1}{4} - \frac{\varepsilon_B}{2})^\top$ and our adjoint PNFFT starts with a three-dimensional decomposition of the truncated torus $\mathbb{T}^3_{\boldsymbol{C}}$. We stress that the usage of the truncated torus $\mathbb{T}^3_{\boldsymbol{C}}$ is crucial in order to avoid severe load balancing problems in this case. If we use a block decomposition of the whole torus $\mathbb{T}^3$ instead, at most an eighth of the processes will receive a nonempty block.

After the parallel forward sort every process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$ owns all the local particles $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$, and the associated particles in the near field radii of these particles, i.e., $\boldsymbol{x}_j$, $j \in \bigcup_{l \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}} \mathcal{I}^{\mathrm{NE}}_{\varepsilon_1}(l)$. Now, the local near field computations (5.3) and (5.5) are performed with a standard linked cell algorithm; see, e.g., [25, Chap. 8.4] or [21, Chap. 3], and the far field computations (5.4) and (5.6) are split into the following three steps.

At first, we compute

$$\hat{a}_{\boldsymbol{k}} := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{j \in \mathcal{M}^{\boldsymbol{s}}_{\boldsymbol{P}}} q_j \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}, \quad \boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}},$$

by an adjoint PNFFT (Algorithm 2). Again, the formal order of summation was chosen in order to reflect the parallel data decomposition. The convolution in frequency domain is a simple pointwise multiplication and is performed locally on each process $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$, i.e.,

$$\hat{d}_{\boldsymbol{k}} := \hat{a}_{\boldsymbol{k}} \hat{R}_{\boldsymbol{k}}, \quad \boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}.$$

Hereby, the Fourier coefficients $\hat{R}_{\boldsymbol{k}}$ of the regularization R are precomputed by a parallel three-dimensional FFT

$$\hat{R}_{\boldsymbol{k}} = \frac{1}{|\mathcal{I}_{\boldsymbol{N}}|} \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{l} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} R(\|\boldsymbol{l} \odot \boldsymbol{N}^{-1}\|_2) \mathrm{e}^{+2\pi \mathrm{i} (\boldsymbol{l} \odot \boldsymbol{N}^{-1}) \boldsymbol{k}}, \quad \boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}. \tag{5.7}$$

Afterward, the far field potentials and fields are computed by a PNFFT (Algorithm 1),

$$\phi^{\mathrm{RF}}_j := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{d}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}, \quad j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}},$$

$$\boldsymbol{E}^{\mathrm{RF}}_j := -\sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{d}_{\boldsymbol{k}} \nabla \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}, \quad j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}.$$

Finally, we use the fine-grained data distribution operation from the parallel sorting library to restore the initial parallel distribution of the nodes $\boldsymbol{x}_j$ together with the computed Coulomb potentials and fields.

In summary we obtain Algorithm 3 for the fast and parallel evaluation of the potentials (5.1) and the fields (5.2). Note that this algorithm can be easily modified for other kernel functions frequently used in the approximation by radial basis functions, e.g., the Gaussian [34] or the (inverse) multiquadric [15] $(x^2 + c^2)^{\pm 1/2}$.

From an abstract point of view the steps of this algorithm are very similar to particle-particle–particle-mesh ($\mathrm{P^3M}$) algorithms [25, Chap. 8]; see also [5] for an overview of particle-mesh algorithms. However, Algorithm 3 differs from Ewald-based particle-mesh methods in several details. At first, the Fourier coefficients of the regularized Coulomb potential must be computed by an FFT within a precomputation step. In contrast, for periodic boundary conditions the Fourier coefficients are given analytically by the well-known Ewald splitting [13]. Furthermore, the fast summation algorithm for nonperiodic boundary conditions makes use of oversampled FFTs and

ALGORITHM 3. PARALLEL, THREE-DIMENSIONAL PARTICLE-PARTICLE–NFFT (P^{2}NFFT) FOR EACH PROCESS $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{P}}$.

Input: $\boldsymbol{N} \in 2\mathbb{N}^3$ multibandwidth,
$\varepsilon_{\mathrm{I}} > 0$ near field cutoff radius,
$\varepsilon_{\mathrm{B}} > 0$ far field regularization border,
nodes $\boldsymbol{x}_j \in \left\{\boldsymbol{x} \in \mathbb{T}^3 : \|\boldsymbol{x}\|_2 \leq \frac{1}{4} - \frac{\varepsilon_{\mathrm{B}}}{2}\right\}$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$,
sources $q_j \in \mathbb{R}$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$.

Precomputation: Compute the Fourier coefficients $\hat{R}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$, given by (5.7) by a parallel, three-dimensional, adjoint FFT.

1: Assign the local nodes $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}} \cup \bigcup_{l \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}} \mathcal{I}^{\mathrm{NE}}_{\varepsilon_{\mathrm{I}}}(l)$, by a parallel forward sort.

2: For $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$ compute $\hat{a}_{\boldsymbol{k}} := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{j \in \mathcal{M}^{\boldsymbol{s}}_{\boldsymbol{P}}} q_j \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$ by an adjoint PNFFT; see Algorithm 2.

3: For $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{r}}_{\boldsymbol{N},\boldsymbol{P}}$ compute the products $\hat{d}_{\boldsymbol{k}} := \hat{a}_{\boldsymbol{k}} \hat{R}_{\boldsymbol{k}}$.

4: For $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$ compute the far field sums

$$\phi^{\mathrm{RF}}_j := \sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{d}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j},$$

$$\boldsymbol{E}^{\mathrm{RF}}_j := -\sum_{\boldsymbol{s} \in \mathcal{P}_{\boldsymbol{P}}} \sum_{\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{s}}_{\boldsymbol{N},\boldsymbol{P}}} \hat{d}_{\boldsymbol{k}} \nabla \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$$

by a PNFFT; see Algorithm 1.

5: For $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$ compute the near field sums

$$\phi^{\mathrm{NE}}_j = R(0) + \sum_{l \in \mathcal{I}^{\mathrm{NE}}_{\varepsilon_{\mathrm{I}}}(j)} q_l \left(\frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2} - R(\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2) \right),$$

$$\boldsymbol{E}^{\mathrm{NE}}_j = \sum_{l \in \mathcal{I}^{\mathrm{NE}}_{\varepsilon_{\mathrm{I}}}(j)} q_l \frac{\boldsymbol{x}_j - \boldsymbol{x}_l}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2} \left(\frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2^2} + R'(\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2) \right)$$

by a linked cell algorithm; see [25, Chap. 8.4] or [21, Chap. 3].

6: For $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$ compute the near field corrections of the potentials $h_j = \phi^{\mathrm{NE}}_j + \phi^{\mathrm{RF}}_j$ and the fields $-\nabla h_j = \boldsymbol{E}^{\mathrm{NE}}_j + \boldsymbol{E}^{\mathrm{RF}}_j$.

7: Restore the initial data distribution by a parallel backward sort.

Output: Approximate potentials $h_j \approx \phi_j$ and fields $-\nabla h_j \approx \boldsymbol{E}_j$, $j \in \mathcal{M}^{\boldsymbol{r}}_{\boldsymbol{P}}$.

rescaling of the particle coordinates into a box of roughly half the size of the initial box. We have seen that these two steps imply load balancing problems for the parallelization of the fast summation algorithm that are not apparent in Ewald-based particle-mesh algorithms. Therefore, Algorithm 3 applies parallel pruned FFTs in order to reduce the load imbalances. Furthermore, we show in Appendix A that the use of oversampled FFTs does not increase the runtime in comparison to P^{3}M algorithms with the same mesh size, since we do not need all the FFT outputs. Even more, for the parameter settings of our following numerical tests the overall runtime of the oversampled FFT with pruned output (4.2) is less than the runtime of a nonpruned

FFT of size $\boldsymbol{N}$. In the following runtime measurement, we will also see that the FFTs take only between 18% and 25% of the total runtime of Algorithm 3. Therefore, the influence of the FFT runtime on the total runtime of Algorithm 3 is rather small.

Following the naming scheme of the P^{3}M, our proposed Algorithm 3 is called particle-particle–NFFT (P^{2}NFFT), since the short range particle-particle interactions are computed in the same way as in P^{3}M algorithms, while the long range particle-mesh part is computed by NFFTs.

6. Numerical results. We implemented Algorithm 1 (PNFFT), Algorithm 2 (adjoint PNFFT), and Algorithm 3 (P^{2}NFFT) and investigated the strong scaling behavior of our implementations on a BlueGene/P architecture. The PNFFT algorithms have been published in the PNFFT software library [44] and the implementation of P^{2}NFFT is part of the publicly available ScaFaCoS software library [54]. In this section we first present the parallel runtime measurements of the parallel NFFT and its adjoint. Second, we show the strong scaling of the P^{2}NFFT. We performed the tests on a BlueGene/P architecture in Research Center Jülich (JuGene) [28]. One node of a BlueGene/P consists of 4 IBM PowerPC 450 cores that run at 850 MHz. These 4 cores share 2 GB of main memory. Therefore, we have 0.5 GB RAM per core, whenever all the cores per node are used. The nodes are connected by a 3D torus network with 425 MB/s bandwidth per link. In total JuGene consists of 73728 nodes, i.e., 294912 cores. Our software has been built with the IBM XL C/C++ compiler (Advanced Edition for Blue Gene/P, V9.0) and the compiler flags `CFLAGS="-O3 -qmaxmem=-1 -qarch=450 -qtune=450"`. As a test system we use a cubic box filled with 12960 particles of a silica melt. It was generated by a simulation of a melting silica crystal using the potential given in [2]. This particle system consists of positive and negative charged ions which are sufficiently homogeneously distributed. We duplicate the initial test system of 12960 particles for 4, 9, and 20 times in every direction of space in order to generate a cubic box filled with 829440, 9447840, and 103680000 particles, respectively. The sum of all charges is equal to zero since the initial sample of 12960 particles is neutrally charged.

6.1. Some notes on performance optimization. Before we present our numerical results, we introduce some performance optimizations that we employed in order to improve the run times of Algorithms 1, 2, and 3. The proposed algorithms have been implemented on top of the well-optimized FFTW software library [19, 18] and its extension to massively parallel architectures called PFFT [43, 45]. Similar to FFTW, our implementation splits the workflow of the algorithm into two stages: First, a possibly time consuming planning part and, second, the fast and optimized computation of the given problem. This concept makes it possible to implement rather flexible algorithms that offer high and portable performance. Beside the use of FFTW, which offers a kind of automatic hardware adaptivity, no effort was spent on compiler or hardware specific optimizations. However, we were able to benefit from the experiences that we gained during the development and implementation of the mature NFFT software library for computing NFFTs [30, 31] that also comes with the serial predecessor of the presented P^{2}NFFT Algorithm 3.

Precomputation of the window function. In order to reduce the computational cost of the evaluation of the window function in Algorithms 1 and 2, we use tensor structure based precomputation and interpolation from lookup tables [31]. Therefore, we reduce the computation of the $(2m+1)^3$ window function values per node in (4.4) to $3(2m+1)$ interpolations of the one-dimensional window function ψ and

$(2m+1)^3$ multiplications in order to compute the tensor products $\varphi(\boldsymbol{x} - \boldsymbol{l} \odot \boldsymbol{n}^{-1}) = \psi(x_0 - \frac{l_0}{n_0})\psi(x_1 - \frac{l_1}{n_1})\psi(x_2 - \frac{l_2}{n_2})$. Note, that the computing time of the window function with tensor product based interpolation does not depend on the particular choice of the window function. In our implementation the flags `PRE_LIN_PSI`, `PRE_QUAD_PSI` and `PRE_KUB_PSI` enable linear, quadratic, and cubic interpolation of the one-dimensional window function $\psi(x)$. Additionally, the flag `PRE_PHI_HAT` enables the precomputation of the Fourier coefficients $\hat{\varphi}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^r_{\boldsymbol{N},\boldsymbol{P}}$. Hereby, we do not store the full set of $|\mathcal{I}_{\boldsymbol{N}}|$ Fourier coefficients. Instead, we exploit the tensor structure of the three-dimensional window function $\hat{\varphi}_{\boldsymbol{k}} = \hat{\psi}_{k_0}\hat{\psi}_{k_1}\hat{\psi}_{k_2}$, $\boldsymbol{k} = (k_0, k_1, k_2)^\top \in \mathcal{I}_{\boldsymbol{N}}$ and store the precomputed $N_0 + N_1 + N_2$ Fourier coefficients of the one-dimensional window functions $\hat{\psi}_{k_t}$, $k_t = -N_t/2, \ldots, N_t/2 - 1$, $t = 0, 1, 2$. Therefore, the evaluation of the three-dimensional Fourier coefficients requires $2|\mathcal{I}_{\boldsymbol{N}}|$ multiplications. For our numerical experiments we chose the Kaiser–Bessel window function with cubic interpolation from precomputed lookup tables.

Transposed FFT output. The PFFT software library employs a parallel FFT algorithm that contains several data transpositions in order to enable the computation of one-dimensional local FFTs [45]. On default these transpositions are reverted after the computation of the FFT, which doubles the amount of global communication. However, a convolution in Fourier space corresponds to a simple pointwise multiplication of two arrays. As long as both arrays are distributed in the same way, a pointwise multiplication can be easily implemented for any parallel data decomposition. Therefore, we omit the additional communication and use the transpositions of the second FFT to restore the initial data decomposition in the following way. First, we compute the parallel adjoint FFT step 3 of the adjoint PNFFT Algorithm 2 with transposed output. Afterward, the convolutions in Fourier space of the adjoint PNFFT (step 4 of Algorithm 2), the P^{2}NFFT (step 3 of Algorithm 3), and the PNFFT (step 1 of Algorithm 1) are computed on transposed arrays. Finally, we compute the parallel FFT step 2 of Algorithm 1 with transposed input and end up with the initial domain decomposition.

Interpolation of the regularization. The computation of the near field step 5 of the P^{2}NFFT Algorithm 3 requires the repetitive evaluation of the regularization $R(r)$ and its derivative $R'(r)$ for $0 \le r \le \varepsilon_I$. Since these two functions are smooth, we use cubic interpolation from precomputed lookup tables to speed up their evaluation.

6.2. Runtimes of PNFFT and adjoint PNFFT. In Figure 6.1 we show the wall clock time measurements of Algorithm 1 (PNFFT) and Algorithm 2 (adjoint PNFFT) for 512^3 Fourier coefficients and 829440 nonequispaced nodes up to 16384 cores of a BlueGene/P architecture. For comparison purposes we show the perfect strong scaling times (perfect) of the first recorded time. In addition, we add the wall clock time of the most time consuming parts of these algorithms. These are the convolution steps 4 and 5 (B, ∇B), the ghost cell communication step 3 (ghost), the FFT step 2 (F), and the deconvolution step 1 (D) of Algorithm 1 and their adjoint counterparts of Algorithm 2, i.e., the adjoint convolution step 1 (adjoint B), the adjoint ghost cell communication step 2 (adjoint ghost), the adjoint FFT step 3 (adjoint F), and the adjoint deconvolution step 4 (adjoint D). Both plots are scaled equally such that a direct comparison of the time measurement between the two algorithms is possible.

The deconvolution step 1 of Algorithm 1 (D) and the adjoint counterpart step 4 of Algorithm 2 (adjoint D) scale perfectly and only represent a small amount of

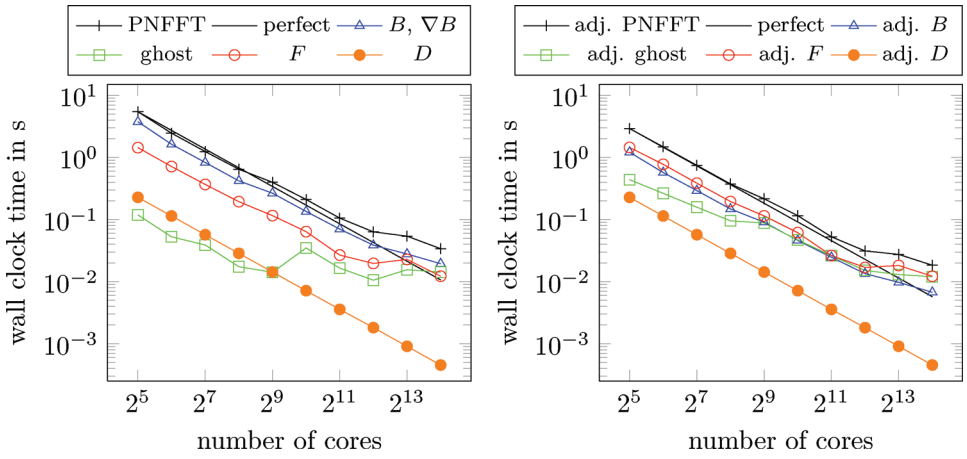

FIG. 6.1. *Wall clock time measurements of Algorithm* 1 *(PNFFT) on the left and Algorithm* 2 *(adjoint PNFFT) on the right with number of nonequispaced nodes* $M = 829440$*, pruned FFT input size* $\boldsymbol{N} = (512, 512, 512)^\top$*, oversampled FFT size* $\boldsymbol{n} = (576, 576, 576)^\top$*, pruned FFT output size* $\boldsymbol{L} = (174, 174, 174)^\top$*, and window cutoff parameter* $m = 4$*.*

the overall run times. The FFT step 2 of Algorithm 1 (F) and its adjoint step 3 of Algorithm 2 (adjoint F) show good strong scaling up to 4096 cores, which corresponds to one rack of the BlueGene/P. We observe a performance penalty for computing the parallel FFT on more than one rack. However, note that the pruned FFT output is only of size 174^3. The FFT-internal two-dimensional distribution of 174^3 complex numbers on 128^2 processes results in very small workload per process.

The discrete convolution step of Algorithm 1 ($B, \nabla B$) includes the calculation of the function values (step 4 of Algorithm 1) and the calculation of the gradients (step 5 of Algorithm 1). Therefore, it is more time consuming than the corresponding adjoint step 1 of Algorithm 2 (adjoint B). Both show good strong scaling behavior. Note that every process gets only 51 nodes $\boldsymbol{x}_j$ in average, if we use 16384 processes in total.

Instead, the ghost cell communication shows bad strong scaling for more than 512 processes. The adjoint ghost cell communication is more expensive than plain ghost cell communication since it additionally involves the computation and synchronization of the partial sums of ghost cell summands. For 16384 processes the adjoint ghost cell communication turns out to be the most time consuming part of the adjoint PNFFT. We observe that the ghost cell communication is the most limiting factor for strong scaling. This slightly improves for larger test cases, where the ratio between the ghost cell communication and the overall computing time decreases.

In Figure 6.2 we show the wall clock time measurements of Algorithm 1 (PNFFT) and Algorithm 2 (adjoint PNFFT) for a medium sized test case with 1024^3 Fourier coefficients and 9447840 nonequispaced nodes. Furthermore, we present in Figure 6.3 the wall clock time measurements of Algorithm 1 (PNFFT) and Algorithm 2 (adjoint PNFFT) for a large test case with 2048^3 Fourier coefficients and 103680000 nonequispaced nodes. Both test cases have been run on up to 65536 cores of a BlueGene/P architecture.

There the computing time of the discrete convolution step, the adjoint FFT, and the adjoint ghost cell communication are nearly the same. Also the strong scaling behavior of all three steps is good. Although the ghost cell communication does not

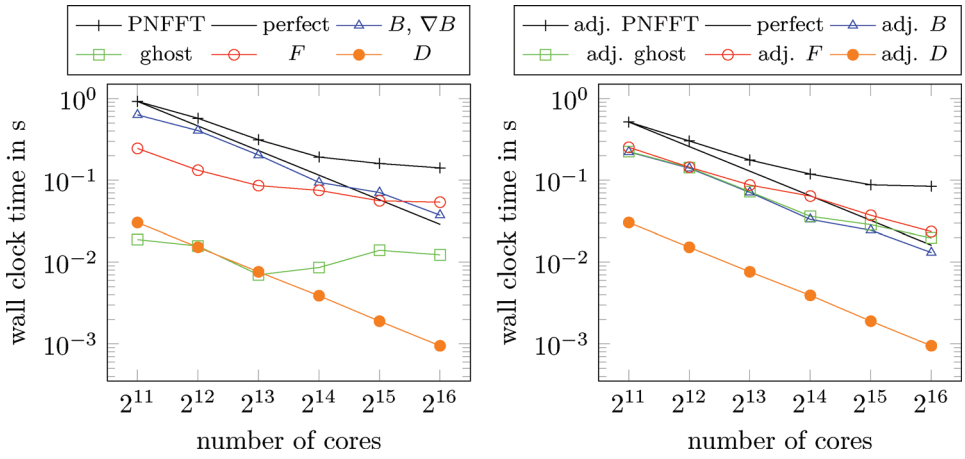

FIG. 6.2. *Wall clock time measurements of Algorithm* 1 *(PNFFT) on the left and Algorithm* 2 *(adjoint PNFFT) on the right with number of nonequispaced nodes* $M = 9447840$*, pruned FFT input size* $\boldsymbol{N} = (1024, 1024, 1024)^\top$*, oversampled FFT size* $\boldsymbol{n} = (1152, 1152, 1152)^\top$*, pruned FFT output size* $\boldsymbol{L} = (340, 340, 340)^\top$*, and window cutoff parameter* $m = 4$*.*

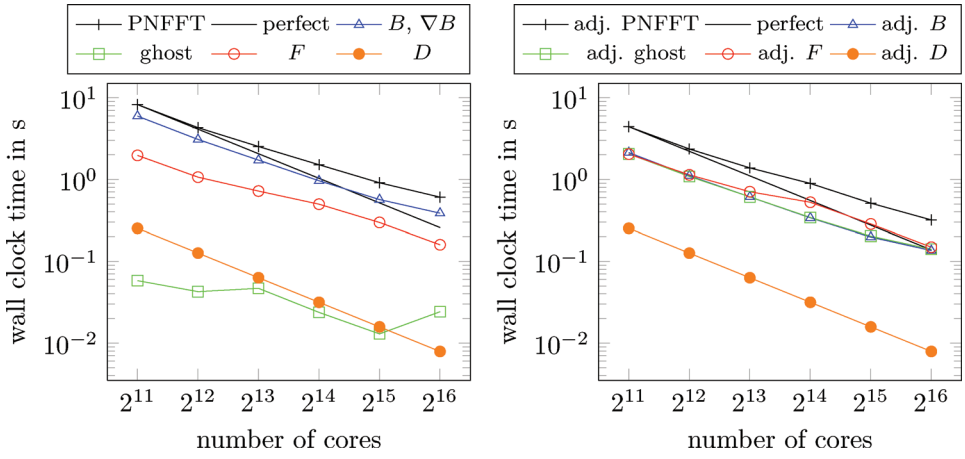

FIG. 6.3. *Wall clock time measurements of Algorithm* 1 *(PNFFT) on the left and Algorithm* 2 *(adjoint PNFFT) on the right with number of nonequispaced nodes* $M = 103680000$*, pruned FFT input size* $\boldsymbol{N} = (2048, 2048, 2048)^\top$*, oversampled FFT size* $\boldsymbol{n} = (2304, 2304, 2304)^\top$*, pruned FFT output size* $\boldsymbol{L} = (674, 674, 674)^\top$*, and window cutoff parameter* $m = 4$*.*

provide good scaling behavior, it only takes 4% of the overall PNFFT run time with 65536 processes. Once more, the deconvolution steps show perfect strong scaling. The parallel pruned FFT shows a performance penalty at 8192 and 16384 processes. Presumably this comes from the fact, that the underlying two-dimensional process grid exceeds the physical dimensions of one rack along the first dimension for 8192 processes and along the first and second dimension for more than 8192 processes. We observe a good scaling of the overall wall clock time of the PNFFT and the adjoint PNFFT that reflects the good scaling of all their most time consuming steps.

Note, that the pruned FFT output sizes used in these examples are not divisible by the number of processes. Therefore, the block decomposition of the parallel FFT does not result in equal blocks and gives rise to load imbalances. These load imbalances

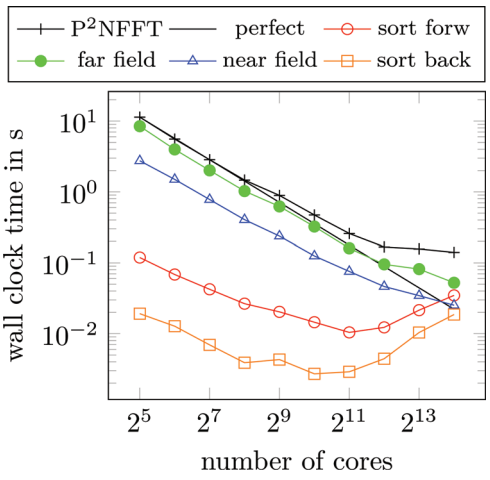

FIG. 6.4. *Algorithm 3 (P^2NFFT) for a silica melt with number of nonequispaced nodes $M = 829440$, rms-potential error $\varepsilon_{\text{pot}} = 10^{-5}$, regularization parameters $\varepsilon_I = \varepsilon_B = 0.0078$, pruned FFT input size $\boldsymbol{N} = (512, 512, 512)^\top$, oversampled FFT size $\boldsymbol{n} = (576, 576, 576)^\top$, pruned FFT output size $\boldsymbol{L} = (174, 174, 174)^\top$, and window cutoff parameter $m = 4$.*

become especially obvious for small FFT output sizes $\boldsymbol{L}$ and large process counts. Therefore, the parallel FFTs of the medium sized 9447840 particle test case presented in Figure 6.2 show underwhelming strong scaling behavior, whereas the parallel FFTs corresponding to the 103680000 particle test case presented in Figure 6.3 show a nearly perfect scaling behavior.

6.3. Runtimes of P²NFFT. Let $\phi_{\text{P}^2\text{NFFT}}(\boldsymbol{x}_j)$, $j = 1, \ldots, M$, denote the potentials that are calculated by Algorithm 3 and $\phi_{\text{REF}}(\boldsymbol{x}_j)$, $j = 1, \ldots, M$, the potentials that are calculated by any high accuracy reference method. For small numbers of particles we chose the direct summation as reference method, while a fast method with higher accuracy was used to calculate the reference potentials for large systems. The relative root mean square (rms) potential error is given by

$$\varepsilon_{\text{pot}} := \left(\sum_{j=1}^{M} |\phi_{\text{REF}}(\boldsymbol{x}_j) - \phi_{\text{P}^2\text{NFFT}}(\boldsymbol{x}_j)|^2 \right)^{1/2} \left(\sum_{j=1}^{M} |\phi_{\text{REF}}(\boldsymbol{x}_j)|^2 \right)^{-1/2}.$$

For the following run time measurements we tuned the parameters of P²NFFT in order to hold $\varepsilon_{\text{pot}} < 10^{-5}$.

In Figure 6.4 we show the wall clock time measurements of Algorithm 3 (P²NFFT) for the silica melt test case with 829440 particles on a BlueGene/P architecture using up to 16384 cores. For comparison purposes we show the perfect strong scaling time (perfect) of the first recorded time. In addition, we add the wall clock time of the most time consuming parts of this algorithm. These are the forward sorting step 1 (sort forw), the far field computation steps 2, 3 and 4 (far field), the near field computation step 5 (near field) and the backward sorting step 7 (sort back) of the P²NFFT Algorithm 3. Note, that the individual wall clock time measurements of the adjoint PNFFT step 2 (adjoint PNFFT) and the PNFFT step 4 (PNFFT) are given in Figure 6.1. We stress that the pruned FFT output size $\boldsymbol{L} = (174, 174, 174)^\top$ is significantly smaller than the oversampled FFT size $\boldsymbol{n} = (576, 576, 576)^\top$ for this

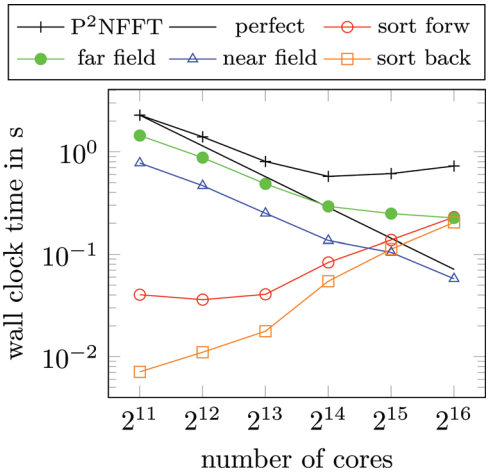

FIG. 6.5. *Algorithm 3 (P²NFFT) for a silica melt with number of nonequispaced nodes* $M = 9447840$*, rms-potential error* $\varepsilon_{\text{pot}} = 10^{-5}$*, regularization parameters* $\varepsilon_I = \varepsilon_B = 0.0039$*, pruned FFT input size* $\boldsymbol{N} = (1024, 1024, 1024)^\top$*, oversampled FFT size* $\boldsymbol{n} = (1152, 1152, 1152)^\top$*, pruned FFT output size* $\boldsymbol{L} = (340, 340, 340)^\top$*, and window cutoff parameter* $m = 4$*.*

test case, i.e., only 2.8% of the oversampled FFT output is necessary to compute the convolution steps of the PNFFT and its adjoint. This problem arises from the fact that we must scale all the nonequispaced nodes in order to fulfill $\|\boldsymbol{x}_j\|_2 < \frac{1}{4} - \frac{\varepsilon_B}{2}$ for all $j = 1, \ldots, M$. However, our algorithm takes care of this fact since we apply a parallel pruned FFT and use the data decomposition of the truncated Torus $\mathbb{T}^3_{\boldsymbol{C}}$ instead of the full Torus $\mathbb{T}^3$; see also Figure 4.2 for details.

Although the near field and far field computations show good strong scaling, we observe that the sorting steps are the most limiting factor for strong scaling of the P²NFFT algorithm. Remember, that the parallel sorting assures the block domain decomposition and generates the ghost cell particles that are necessary for the near field computations. Since the parallel sorting algorithm calls `MPI_Alltoallv` we see the typical almost linear increase of sorting time from 2048 to 16384 processes. However, we stress that using 16384 cores for this test case implies a very small number of 51 local nodes for every process. Therefore, generating the ghost particles that are necessary for the near field computation is rather costly in comparison to the actual near field computations itself.

Note, that typical molecular dynamics (MD) software packages already have a high performance implementation of the near field part and use a blockwise domain decomposition that perfectly fits the three-dimensional data decomposition of our P²NFFT Algorithm 3. In this case we could redirect the near field computation steps 5 and 6 of our P²NFFT Algorithm 3 to the MD software package and omit the whole sorting steps 1 and 7 of Algorithm 3. Then we are left with the stand-alone far field part without sorting, e.g., steps 2, 3, and 4 of Algorithm 3. The corresponding run time behavior of the far field part is dominated by the parallel nonequispaced Fourier transforms, which were analyzed in detail in Figure 6.1, Figure 6.2 and Figure 6.3.

In Figure 6.5 we show the wall clock time measurements of Algorithm 3 (P²NFFT) for the silica melt test case with 9447840 particles on a BlueGene/P architecture using up to 65536 cores. Finally, in Figure 6.6 we see the wall clock time measurements of Algorithm 3 (P²NFFT) for the silica melt test case with 103680000 particles on a BlueGene/P architecture using up to 65536 cores. A comparison of Figures 6.5 and

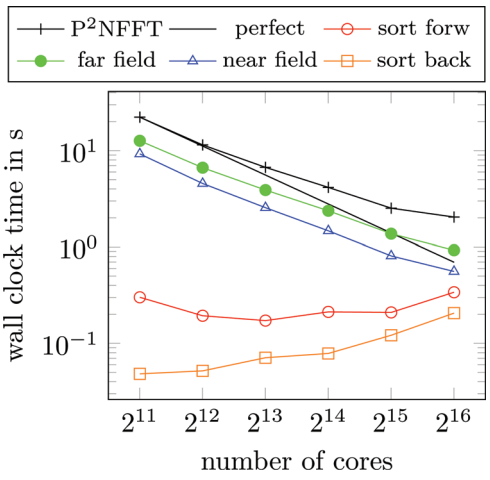

FIG. 6.6. *Algorithm 3 (P^2NFFT) for a silica melt with number of nonequispaced nodes $M = 103680000$, rms-potential error $\varepsilon_{\text{pot}} = 10^{-5}$, regularization parameters $\varepsilon_I = \varepsilon_B = 0.002$, pruned FFT input size $\boldsymbol{N} = (2048, 2048, 2048)^\top$, oversampled FFT size $\boldsymbol{n} = (2304, 2304, 2304)^\top$, pruned FFT output size $\boldsymbol{L} = (674, 674, 674)^\top$, and window cutoff parameter $m = 4$.*

6.6 yields a better scaling of the larger test case since the quota of sorting decreases. In both test cases the wall clock time of the sorting steps increases almost linearly with the number of processes and prevents a good scaling of the overall run time. However, we stress again that the number of local particles is rather small in comparison to the number of used processes in these test cases and typical MD software may omit the sorting by using an appropriate domain decomposition.

7. Conclusions. In this paper we have presented new parallel algorithms for computing the NFFT and its adjoint on distributed memory architectures. These algorithms have been implemented and published in the PNFFT [44] software library. To our knowledge this is the first publicly available massively parallel NFFT implementation. Run time tests on a BlueGene/P system using up to 65536 cores showed a good scalability of our implementation. Furthermore, we applied the parallel nonequispaced Fourier transform in order to derive a parallel fast summation algorithm (P^2NFFT). This is the first time a particle-mesh algorithm was applied in order to compute long ranged interactions with nonperiodic boundary conditions in parallel. We implemented the P^2NFFT algorithm within the ScaFaCoS [54] software library and presented scaling results on a BlueGene/P system using up to 65536 cores. The run time behavior was dominated by the good scaling of the PNFFT-driven far field computations, the near field computations, and the parallel particle sorting that we introduced in order to get a flexible algorithm independent of the incoming domain decomposition. However, the sorting can be circumvented by using a common blockwise domain decomposition.

Appendix A. Complexity estimation of pruned FFT. For the sake of simplicity we assume a cubic Fourier mesh, i.e., $N := N_0 = N_1 = N_2$, $n := n_0 = n_1 = n_2$, $L := L_0 = L_1 = L_2$. In this section, we compare the complexity of an ordinary three-dimensional FFT of total size N^3 and the pruned three-dimensional FFT (3.3) of total size n^3 with pruned FFT input size N^3 and pruned FFT output size L^3. Let $\sigma := n/N$ and $\tau := L/N$. The three-dimensional pruned FFT (3.3) is computed via a typical row-column algorithm, i.e., three successive sets of one-dimensional FFTs.

Before each one-dimensional FFT, the one-dimensional input arrays of length N are padded with zeros up to length n. Afterward, only L outputs of each one-dimensional FFT are kept for the next calculations. In the following, we count the number of arithmetic operations but ignore the time needed for any data movement.

The first step consists of N^2 FFTs of size n at the total cost of

$$c_0 = C_{\mathrm{FFT}} \cdot N^2 n \log n = C_{\mathrm{FFT}} \cdot \sigma N^3 (\log N + \log \sigma)$$

arithmetic operations. Hereby, C_{FFT} is a constant that accounts for the arithmetic operations for each calculation of a butterfly step. During the second step NL FFTs of size n are computed at the total cost of

$$c_1 = C_{\mathrm{FFT}} \cdot NLn \log n = C_{\mathrm{FFT}} \cdot \sigma\tau N^3 (\log N + \log \sigma)$$

arithmetic operations. Finally, L^2 FFTs of size n are computed at the total cost of

$$c_2 = C_{\mathrm{FFT}} \cdot L^2 n \log n = C_{\mathrm{FFT}} \cdot \sigma\tau^2 N^3 (\log N + \log \sigma)$$

arithmetic operations. In summary, the complexity of the three-dimensional pruned FFT is given by

$$c_{\sigma,\tau} := c_0 + c_1 + c_2 = C_{\mathrm{FFT}} \cdot \sigma \left(1 + \tau + \tau^2\right) N^3 (\log N + \log \sigma).$$

Note, that the well-known complexity of a nonpruned three-dimensional FFT of size N^3 is given by

$$c_{1,1} = C_{\mathrm{FFT}} \cdot 3N^3 \log N$$

and the complexity of a three-dimensional oversampled FFT with oversampling factor σ is given by

$$c_{\sigma,\sigma} = C_{\mathrm{FFT}} \cdot \sigma(1 + \sigma + \sigma^2) N^3 (\log N + \log \sigma).$$

Therefore, we get an estimation of the increase of run time due to the pruned FFT,

$$\frac{c_{\sigma,\tau}}{c_{1,1}} = \frac{\sigma(1 + \tau + \tau^2)}{3} \left(1 + \frac{\log \sigma}{\log N}\right) \approx \frac{\sigma(1 + \tau + \tau^2)}{3}.$$

For a rather conservative oversampling factor of $\sigma = 2$ and nonpruned output ($\tau = 2$) the increase of run time during the FFT would be about $c_{2,2}/c_{1,1} \approx 4.67$. However, if we set $\sigma = 1.125$ and $\tau = 0.34$ according to the run time measurements given in Figure 6.4 we get $c_{1.125,0.34}/c_{1,1} \approx 0.546$, i.e., computing only 174^3 outputs of an oversampled three-dimensional FFT of size 576^3 is even faster than computing all outputs of an ordinary three-dimensional FFT of size 512^3.

Acknowledgments. We thank the anonymous reviewers for their helpful suggestions. We are grateful to the Jülich Supercomputing Center for providing the computational resources on Jülich BlueGene/P (JuGene). We wish to thank Dr. Franz Ghler, who supplied the silica melt test case. Furthermore, we gratefully acknowledge the help of Dr. Michael Hofmann on the parallel sorting algorithms and the help of Rene Halver on the implementation of the parallel linked cell algorithm.

REFERENCES

[1] O. Ayala and L.P. Wang, *Parallel implementation and scalability analysis of 3D fast Fourier transform using 2D domain decomposition*, Parallel Comput., 39 (2013), pp. 58–77.

[2] B.W.H. van Beest and G.J. Kramer, *Force fields for silicas and aluminophosphates based on ab initio calculations*, Phys. Rev. Lett., 64 (1990), pp. 1955–1958.

[3] G. Beylkin, *On the fast Fourier transform of functions with singularities*, Appl. Comput. Harmon. Anal., 2 (1995), pp. 363–381.

[4] H. Dachsel, M. Hofmann, and G. Rünger, *Library support for parallel sorting in scientific computations*, in Proceedings of the 13th International Euro-Par Conference, 4641, Lecture Notes in Comput. Sci. 2007, Springer, Berlin, pp. 695–704.

[5] M. Deserno and C. Holm, *How to mesh up Ewald sums.* I. *A theoretical and numerical comparison of various particle mesh routines*, J. Chem. Phys., 109 (1998), pp. 7678–7693.

[6] H.Q. Ding, R.D. Ferraro, and D.B. Gennery, *A portable 3d FFT package for distributed-memory parallel architectures*, in Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, 1995, pp. 70–71,

[7] A.J.W. Duijndam and M. A. Schonewille, *Nonuniform fast Fourier transform*, Geophysics, 64 (1999), pp. 539–551.

[8] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14 (1993), pp. 1368–1393.

[9] H. Eggers, T. Knopp, and D. Potts, *Field inhomogeneity correction based on gridding reconstruction*, IEEE Trans. Med. Imaging, 26 (2007), pp. 374–384.

[10] B. Elbel and G. Steidl, *Fast Fourier transform for nonequispaced data*, in Approximation Theory IX, C.K. Chui and L.L. Schumaker, eds., Vanderbilt University Press, Nashville, TN, 1998, pp. 39–46.

[11] M. Eleftheriou, J.E. Moreira, B.G. Fitch, and R.S. Germain, *A volumetric FFT for BlueGene/L*, High Performance Computing, Lecture Notes in Comput. Sci. 2913, T.M. Pinkston and V.K. Prasanna, eds., Springer, Berlin, 2003, pp, 194–203.

[12] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, and L.G. Pedersen, *A smooth particle mesh Ewald method*, J. Chem. Phys., 103 (1995), pp. 8577–8593.

[13] P.P. Ewald, *Die Berechnung optischer und elektrostatischer Gitterpotentiale*, Ann. Phys. (4), 369 (1921), pp. 253–287.

[14] B. Fang, Y. Deng, and G. Martyna, *Performance of the 3D FFT on the 6D network torus QCDOC parallel supercomputer*, Comput. Phys. Comm., 176 (2007), pp. 531–538.

[15] M. Fen and G. Steidl, *Fast NFFT based summation of radial functions*, Sampl. Theory Signal Image Process., 3 (2004), pp. 1–28.

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

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

[18] M. Frigo and S.G. Johnson, *The design and implementation of FFTW3*, Proc. IEEE, 93 (2005), pp. 216–231.

[19] M. Frigo and S.G. Johnson, *FFTW, C Subroutine Library*, http://www.fftw.org (2009).

[20] L. Greengard and J.-Y. Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Rev., 46 (2004), pp. 443–454.

[21] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical simulation in molecular dynamics*, Texts Comput. Sci. Eng. 5, Springer, Berlin, 2007.

[22] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2: Advanced Features of the Message-Passing Interface*, MIT Press, Cambridge, MA, 1999.

[23] F. Hedman and A. Laaksonen, *Ewald summation based on nonuniform fast Fourier transform*, Chem. Phys. Lett., 425 (2006), pp. 142–147.

[24] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, *GROMACS 4 : Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation*, J. Chem. Theory Comput., 4 (2008), pp. 435–447.

[25] R.W. Hockney and J.W. Eastwood, *Computer Simulation Using Particles*, Taylor & Francis, Bristol, PA, 1988.

[26] M. Hofmann and G. Rünger, *Fine-grained data distribution operations for particle codes*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, 16th European PVM/MPI Users Group Meeting, Lecture Notes in Comput. Sci. 5659, M. Ropo, J. Westerholm, and J. Dongarra, eds., Springer, 2009, pp. 54–63.

[27] 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. Imaging, 10 (1991), pp. 473–478.

[28] JuGene: Jülich Blue Gene/P, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUGENE/JUGENE_node.html.
[29] I. Kabadshow and H. Dachsel, *The error-controlled fast multipole method for open and periodic boundary conditions*, in Fast Methods for Long-Range Interactions in Complex Systems, IAS Series 6, G. Sutmann, P. Gibbon, and T. Lippert, eds., Forschungszentrum Jülich, Jülich, 2011, pp. 85–113.
[30] J. Keiner, S. Kunis, and D. Potts, *NFFT* 3.0*, C Subroutine Library*, http://www.tu-chemnitz.de/~potts/nfft.
[31] 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), pp. 1–30.
[32] S. Kunis and S. Kunis, *The nonequispaced FFT on graphics processing units*, PAMM, 12 (2012), pp. 7–10.
[33] S. Kunis and D. Potts, *Stability results for scattered data interpolation by trigonometric polynomials*, SIAM J. Sci. Comput., 29 (2007), pp. 1403–1419.
[34] S. Kunis, D. Potts, and G. Steidl, *Fast Gauss transform with complex parameters using NFFTs*, J. Numer. Math., 14 (2006), pp. 295–303.
[35] N. Li, 2*DECOMP&FFT, Parallel FFT Subroutine Library*, http://www.2decomp.org.
[36] N. Li and S. Laizet, 2*DECOMP & FFT - A Highly Scalable* 2*D Decomposition Library and FFT Interface*, Cray User Group 2010 Conference, Edinburgh, Scotland, 2010, pp. 1–13.
[37] H.J. Limbach, A. Arnold, B.A. Mann, C. Holm, and G. Berne, *ESPResSo — an extensible simulation package for research on soft matter systems*, Comput. Phys. Comm., 174 (2006), pp. 704–727.
[38] T. MacFarland, H. Couchman, F. Pearce, and J. Pichlmeier, *A new parallel code for very large-scale cosmological simulations*, New Astron. Rev., 3 (1998), pp. 687–705.
[39] MPI Forum, *MPI: A Message-Passing Interface Standard. Version* 2.2, http://www.mpi-forum.org (2009).
[40] D. Pekurovsky, *P3DFFT, Parallel FFT Subroutine Library*, http://code.google.com/p/p3dfft.
[41] D. Pekurovsky, *P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions*, SIAM J. Sci. Comput., 34 (2012), pp. C192–C209.
[42] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kalé, and K. Schulten, *Scalable molecular dynamics with NAMD*, J. Comput. Chem., 26 (2005), pp. 1781–1802.
[43] M. Pippig, *PFFT, Parallel FFT Subroutine Library*, http://www.tu-chemnitz.de/~mpip/software.php (2011).
[44] M. Pippig, *PNFFT, Parallel Nonequispaced FFT Subroutine Library*, http://www.tu-chemnitz.de/~mpip/software.php (2011).
[45] M. Pippig, *PFFT: An extension of FFTW to massively parallel architectures*, SIAM J. Sci. Comput., 35 (2013), pp. C213–C236.
[46] M. Pippig and D. Potts, *Particle simulation based on nonequispaced fast Fourier transforms*, in Fast Methods for Long-Range Interactions in Complex Systems, IAS Series 6, G. Sutmann, P. Gibbon, and T. Lippert, eds., Forschungszentrum Jülich, Jülich, 2011, pp. 131–158.
[47] S.J. Plimpton, *Parallel FFT Subroutine Library*, http://www.sandia.gov/~sjplimp/docs/fft/README.html.
[48] S.J. Plimpton, R. Pollock, and M. Stevens, *Particle-mesh Ewald and rRESPA for parallel molecular dynamics simulations*, in Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, 1997, SIAM, Philadelphia, 1997.
[49] E. Pollock and J. Glosli, *Comments on P3M, FMM, and the Ewald method for large periodic Coulombic systems*, Comput. Phys. Comm., 95 (1996), pp. 93–110.
[50] D. Potts and G. Steidl, *Fast summation at nonequispaced knots by NFFT*, SIAM J. Sci. Comput., 24 (2003), pp. 2013–2037.
[51] D. Potts, G. Steidl, and A. Nieslony, *Fast convolution with radial kernels at nonequispaced knots*, Numer. Math., 98 (2004), pp. 329–351.
[52] D. Potts, G. Steidl, and M. Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, in Modern Sampling Theory: Mathematics and Applications, J.J. Benedetto and P.J.S.G. Ferreira, eds., Birkhäuser, Boston, MA, 2001, pp. 247–270.
[53] D.F. Richards, M.P. Surh, J.A. Gunnels, J.N. Glosli, B. Chan, M.R. Dorr, E.W. Draeger, J.L. Fattebert, W.D. Krauss, T. Spelce, and F.H. Streitz, *Beyond homogeneous decomposition: Scaling long-range forces on massively parallel systems*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM Press, New York, 2009, 60.
[54] ScaFaCoS - Scalable Fast Coloumb Solvers, http://scafacos.github.io.

[55] G. Steidl, *A note on fast Fourier transforms for nonequispaced grids*, Adv. Comput. Math., 9 (1998), pp. 337–353.

[56] D. Takahashi, *An implementation of parallel 3-D FFT with 2-D decomposition on a massively parallel cluster of multi-core processors*, in Parallel Processing and Applied Mathematics, Lecture Notes in Comput. Sci. 6067, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, eds., Springer, Berlin, 2010, pp. 606–614.

[57] T. Theuns, *Parallel P3M with exact calculation of short range forces*, Comput. Phys. Comm., 78 (1994), pp. 238–246.

[58] T. Volkmer, *OpenMP Parallelization in the NFFT Software Library*, Preprint TU Chemnitz, preprint 7, 2012, http://www.tu-chemnitz.de/~potts/paper/openmpNFFT.pdf.

[59] A.F. Ware, *Fast approximate Fourier transforms for irregularly spaced data*, SIAM Rev., 40 (1998), pp. 838–856.