Journal of Computational Physics 278 (2014) 76–91

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A parallel fast multipole method for elliptic difference equations

Sebastian Liska *, Tim Colonius *

Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA

ARTICLE INFO

Article history:
Received 18 September 2013
Received in revised form 3 July 2014
Accepted 29 July 2014
Available online 12 August 2014

Keywords:
Fast multipole method
Fast convolution
Difference equation
Green's function
Infinite domain
Parallel computing
Discrete operator
Elliptic solver

ABSTRACT

A new fast multipole formulation for solving elliptic difference equations on unbounded domains and its parallel implementation are presented. These difference equations can arise directly in the description of physical systems, e.g. crystal structures, or indirectly through the discretization of PDEs. In the analog to solving continuous inhomogeneous differential equations using Green's functions, the proposed method uses the fundamental solution of the discrete operator on an infinite grid, or lattice Green's function. Fast solutions $\mathcal{O}(N)$ are achieved by using a kernel-independent interpolation-based fast multipole method. Unlike other fast multipole algorithms, our approach exploits the regularity of the underlying Cartesian grid and the efficiency of FFTs to reduce the computation time. Our parallel implementation allows communications and computations to be overlapped and requires minimal global synchronization. The accuracy, efficiency, and parallel performance of the method are demonstrated through numerical experiments on the discrete 3D Poisson equation.

1. Introduction

Numerical simulations of physical phenomena often require fast solutions to linear, elliptic difference equations with constant coefficients on regular, unbounded domains. These difference equations naturally arise in the description of physical phenomena including random walks [1], crystal physics [2], and quantum mechanics [3]. Additionally, such difference equations can result from the discretization of PDEs on infinite regular grids or meshes [4–7]. Apart from the accuracy with which the underlying PDE is solved, accurate solution of the difference equations themselves is relevant for compatible spatial discretization schemes that enforce discrete conservation laws [8,9]. Examples of these techniques include finite-volume methods, mimetic schemes, covolume methods, and discrete calculus methods.

The present method considers difference equations formally defined on unbounded Cartesian grids. Solutions to the difference equations are obtained through the convolution of the fundamental solution of the discrete operator with the source terms of the difference equations. As a result, the formally infinite grid can be truncated to a finite computational grid by removing cells that contain negligible source strength. The ease with which this technique is able to adapt the computational domain makes it well-suited for applications involving the temporal evolution of irregular source distributions. For problems that are efficiently described by block-structured grids it is possible adapt the computational domain by simply adding or removing blocks; an example of this technique applied to an incompressible flow is provided in Section 5.

* Corresponding authors.
E-mail addresses: sliska@caltech.edu (S. Liska), colonius@caltech.edu (T. Colonius).

http://dx.doi.org/10.1016/j.jcp.2014.07.048
0021-9991/

The fundamental solution of discrete operators on regular grids, or lattices, are often referred to as lattice Green's functions (LGFs). Expressions for LGFs can be readily obtained in the form of Fourier integrals, but it is typically not possible to reduce the integral representations to expressions only involving a few elementary functions [10,11]. The analytical treatment and the numerical evaluation of many LGFs is facilitated by the availability of asymptotic expansions [12–14]. Although LGFs have been extensively studied, they have rarely been used for solving large systems of elliptic difference equations (exceptions include 2D problems [15,6,7]). The present work extends the use of LGFs to large scale computations involving solutions to 3D elliptic difference equations.

Solving the system of difference equations using LGFs requires evaluating discrete convolutions of the form

$$u(\mathbf{x}_i) = [K * \mathbf{f}](\mathbf{x}_i) = \sum_{j=0}^{M-1} K(\mathbf{x}_i, \mathbf{y}_j) f(\mathbf{y}_j), \quad i = 0, 1, \ldots, N-1, \tag{1}$$

where $K(\mathbf{x}_i, \mathbf{y}_j)$ is the kernel describing the influence of a source located at $\mathbf{y}_j$ with strength $f(\mathbf{y}_j)$ has on the field $u(\mathbf{x})$ at location $\mathbf{x}_i$. For the case of $M = N$, the straightforward approach to evaluate Eq. (1) requires $\mathcal{O}(N^2)$ operations. There are several techniques for evaluating Eq. (1) in $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ operations. A few of these techniques are FMMs, FFT-based methods, particle-in-cell methods, particle-mesh methods, multigrid techniques, multilevel local-correction methods, and hierarchical matrix techniques. In the interest of brevity, a literature review of all the methods related to the fast evaluation of Eq. (1) is omitted; instead we focus our attention on FMMs.

The performance of FMMs relies on the existence of a compressed, or low-rank, representation of the far-field behavior of $K(\mathbf{x}, \mathbf{y})$ that can be used to evaluate Eq. (1) to a prescribed tolerance. Classical fast multipole methods [16,17] require analytical expansions of the far-field behavior of kernels in order to derived low-rank approximations. Although classical FMMs can be developed for the asymptotic expansion of LGFs, alternative FMMs that are better suited for complicated kernel expressions have been developed. Kernel-independent FMMs [18–22,7] do not require analytical expansions of the far-field; instead, for suitable kernels, these methods only require numerical evaluations of the kernel.

The present method is a kernel-independent interpolation-based FMM for non-oscillatory translation-invariant kernels [23,21]. These FMMs achieve low-rank approximations of the kernel by projecting the kernel onto a finite basis of interpolation functions. Interpolation-based FMMs [23,21] use Chebyshev interpolation and accelerate convolutions involving the compressed kernel using singular-value-decompositions (SVDs). In contrast, our method uses polynomial interpolation on equidistant nodes and accelerates convolutions involving the compressed kernel using FFTs. The use of intermediate regular grids and fast FFT-based convolutions have been used by other FMMs [24,25,19,26], and shown to particularly useful in accelerating the computations of 3D methods [19]. The use of intermediate regular grids in our method has the added advantage of simplifying the multilevel algorithm, since sources and evaluation points are defined on Cartesian grids at all levels of the multilevel scheme. The spatial regularity allows for the same fast convolution techniques to be used in determining near-field and far-field contributions. In addition to the base algorithm, our method allows for pre-computations that further accelerate the solver.

The present FMM is similar to the recent 2D FMM [7] in that they both solve difference equations on unbounded domains. In contrast to our method, this method uses skeleton/proxy points and rank-revealing factorizations to obtain low-rank approximations of the kernel. Although we think it is possible to extend this method to 3D, we refrain from speculating on the performance of the algorithm since such extensions are not explored in current literature and their details are unclear to us.

Details regarding LGFs and their relation to solving difference equations on unbounded domains are presented in Section 2. This section also describes methods for performing fast convolutions based on kernel compression and FFT techniques, and presents a context in which these two techniques can combined to yield an even faster convolution scheme. The resulting fast multipole algorithm and its parallel extension are then described in Section 3. Finally, serial and parallel numerical experiments are reported and analyzed in Section 4.

2. Lattice Green's functions and fast block-wise convolution techniques

2.1. Solving difference equations on infinite Cartesian grids

The method proposed in this paper is designed to solve inhomogeneous, linear, constant-coefficient difference equations on unbounded domains. As a representative problem, we consider in detail the difference equations resulting from the discretization of Poisson's equation in 3D. Consider the Poisson equation

$$[\Delta u](\mathbf{x}) = f(\mathbf{x}), \quad supp(f) \in \Omega, \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^3$, Ω is a bounded domain in $\mathbb{R}^3$, and $u(\mathbf{x})$ decays as $1/|\mathbf{x}|$ at infinity. Eq. (2) has the analytic solution

$$u(\mathbf{x}) = [G * f](\mathbf{x}) = \int_{\Omega} G(\mathbf{x} - \boldsymbol{\xi}) f(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{3}$$

where $G(\mathbf{x}) = -1/4\pi|\mathbf{x}|$ is the fundamental solution of the Laplace operator. Discretizing Eq. (2) on an infinite uniform Cartesian grid using a standard second-order finite-difference or finite-volume scheme produces a set of difference equations

$$[\mathsf{Lu}](\mathbf{n}) = \mathsf{f}(\mathbf{n}), \quad supp(\mathsf{f}) \in \Omega_h, \tag{4}$$

where L is the standard 7-pt discrete Laplace operator, $\mathbf{n} \in \mathbb{Z}^3$, and Ω_h is a bounded domain in $\mathbb{Z}^3$. In practice, the constraint on $supp(\mathsf{f})$ can be relaxed by prescribing a finite tolerance and requiring that all non-negligible sources, i.e. sources with a magnitude greater than the prescribed tolerance, be located in a bounded region. The solution to Eq. (4) is given by

$$\mathsf{u}(\mathbf{n}) = [\mathsf{G} * \mathsf{f}](\mathbf{n}) = \sum_{\mathbf{m} \in \Omega_h} \mathsf{G}(\mathbf{n} - \mathbf{m})\mathsf{f}(\mathbf{m}), \tag{5}$$

where $\mathsf{G}(\mathbf{n})$ is the fundamental solution of the discrete Laplace operator. An expression for $\mathsf{G}(\mathbf{n})$ in terms of Fourier integrals is provided by

$$\mathsf{G}(\mathbf{n}) = \frac{1}{8\pi^3} \int\limits_{[-\pi,\pi]^3} \frac{\exp(-i\mathbf{n}\cdot\boldsymbol{\xi})}{2\cos(\xi_1) + 2\cos(\xi_2) + 2\cos(\xi_3) - 6}\, d\boldsymbol{\xi}. \tag{6}$$

The expression in Eq. (6) is readily obtained by first using Discrete Fourier Transforms (DFTs) to diagonalize L. The diagonalized operator is then inverted and subsequently transformed back to the original space using inverse DFTs. Infinite sums in the resulting expression are converted to integrals using appropriate limiting procedures. Details regarding the construction of Eq. (6) and expressions for the fundamental solutions to other discrete operators are found in [1,12,14]. Additionally, Appendix A provides an outline of the numerical procedures used to evaluate $\mathsf{G}(\mathbf{n})$ for small values of $|\mathbf{n}|$.

Although it is possible to compute $\mathsf{G}(\mathbf{n})$ by numerically evaluating Eq. (6), for large values of $|\mathbf{n}|$ it is more efficient to evaluate the LGF via its asymptotic expansion. Techniques for constructing asymptotic expansions of LGFs to arbitrary order are described in [12–14]. Let $A^q_{\mathsf{G}}(\mathbf{x})$ denote the q-term asymptotic expansion of $\mathsf{G}(\mathbf{n})$. For the 3D case, we define $A^q_{\mathsf{G}}(\mathbf{x})$ as the unique rational function such that

$$\mathsf{G}(\mathbf{n}) = A^q_{\mathsf{G}}(\mathbf{n}) + \mathcal{O}\left(|\mathbf{n}|^{-2q-1}\right) \tag{7}$$

as $|\mathbf{n}| \to \infty$. For $q = 2$, this asymptotic expansion is given by

$$A^2_{\mathsf{G}}(\mathbf{x}) = -\frac{1}{4\pi|\mathbf{x}|} - \frac{x_1^4 + x_2^4 + x_3^4 - 3x_1^2x_2^2 - 3x_1^2x_3^2 - 3x_2^2x_3^2}{16\pi|\mathbf{x}|^7}, \tag{8}$$

where $\mathbf{x} = (x_1, x_2, x_3)$. As expected, the first term in Eq. (8) corresponds to the fundamental solution of the Laplace operator. We note that, as is the case for many asymptotic expansions, it is not always possible to increase the accuracy of the expression for a fixed argument by increasing the number of terms.[1]

Despite the fact that $G(\mathbf{x})$ and $\mathsf{G}(\mathbf{n})$ share the same asymptotic behavior, there are significant differences in their behavior near the origin. Unlike $G(\mathbf{x})$, which is singular at the origin, $\mathsf{G}(\mathbf{n})$ is remains finite for all values of $\mathbf{n}$. $G(\mathbf{x})$ is scale-invariant, i.e. there exists a k such that $G(\alpha\mathbf{x}) = \alpha^k G(\mathbf{x})$, whereas $\mathsf{G}(\mathbf{n})$ is not scale-invariant. Furthermore, $G(\mathbf{x})$ is spherical symmetric about the origin, as opposed to $\mathsf{G}(\mathbf{n})$ which has reflectional symmetry about the principal axes and is invariant under index permutations.

In addition to providing expressions for the fast evaluation of LGFs, asymptotic expansions of LGFs allow for the sum given in Eq. (5) to be decomposed into three parts,

$$\mathsf{u}(\mathbf{n}) = \mathsf{u}^{\text{direct}}(\mathbf{n}) + u^{\text{asympt},q}(\mathbf{n}) + \epsilon(\mathbf{n}), \tag{9}$$

where

$$\mathsf{u}^{\text{direct}}(\mathbf{n}) = \sum_{\mathbf{m} \in \Omega_h^{\text{direct}}(\mathbf{n})} \mathsf{G}(\mathbf{n} - \mathbf{m})\mathsf{f}(\mathbf{m}), \tag{10}$$

$$u^{\text{asympt},q}(\mathbf{n}) = \sum_{\mathbf{m} \in \Omega_h \setminus \Omega_h^{\text{direct}}(\mathbf{n})} A^q_{\mathsf{G}}(\mathbf{n} - \mathbf{m})\mathsf{f}(\mathbf{m}), \tag{11}$$

and $\epsilon(\mathbf{n})$ is the error due to approximating $\mathsf{G}(\mathbf{n})$ with $A^q_{\mathsf{G}}(\mathbf{n})$ over the region $\Omega_h \setminus \Omega_h^{\text{direct}}$. The region $\Omega_h^{\text{direct}}(\mathbf{n})$ is a subset of Ω_h for which the LGF is evaluated directly, i.e. via numerical evaluation of Eq. (6), as opposed to being evaluated via its asymptotic expansion. Typically, the region $\Omega_h^{\text{direct}}(\mathbf{n})$ is defined by a small cubic box centered at the grid point $\mathbf{n}$.[2] The first term of Eq. (9), $\mathsf{u}^{\text{direct}}(\mathbf{n})$, is a grid function evaluated at the grid point $\mathbf{n}$, whereas the second term, $u^{\text{asympt},q}(\mathbf{n})$, is a continuous function evaluate at the location of the grid point $\mathbf{n}$. As will be discussed in subsequent sections, this decomposition allows for $u^{\text{asympt},q}(\mathbf{n})$ to be evaluated using fast techniques developed for continuous kernels.

[1] For example, for $|\mathbf{n}| = 10$ the minimum value of $|A^q_{\mathsf{G}}(\mathbf{n}) - \mathsf{G}(\mathbf{n})|/\mathsf{G}(\mathbf{n})$ is approximately 10^{-7} and is achieved by $n = 6$. Increasing or decreasing n, i.e. the number of terms in the asymptotic expansion, increases the relative difference between the $A^q_{\mathsf{G}}(\mathbf{n})$ and $\mathsf{G}(\mathbf{n})$ for $|\mathbf{n}| = 10$.

[2] For the case of the 3D discrete Laplace operator, choosing Ω_h^{direct} to be a cubic box with side lengths of 14, 41, and 134 grid points is sufficient to achieve relative errors less than 10^{-5}, 10^{-10}, and 10^{-15}, respectively, using the five term asymptotic expansion. The size of Ω_h^{direct} can be reduced by including more terms in the asymptotic expansion, for example, a box with side lengths of 38 grid points and the thirteen term asymptotic expansion achieve relative error less than 10^{-15}.

2.2. Fast convolutions on regular grids via FFTs

Although discrete convolutions via FFTs is a well-known technique, a brief description is provided in order to introduce procedures and notation subsequently referenced in different steps of the overall algorithm. Consider the one-dimensional the convolution given by

$$u(x_i) = \sum_{j=0}^{M-1} K(x_i, y_j) f(y_j), \quad i = 0, 1, \ldots, N-1. \tag{12}$$

where $x_i = x_0 + ih$ for $i = 0, 1, \ldots, N-1$, and $y_j = y_0 + jh$ for $j = 0, 1, \ldots, M-1$. If the kernel $K(x, y)$ is translation invariant, i.e. $K(x, y) = K(x - y)$, then Eq. (12) can be expressed as the discrete convolution between two vectors,

$$u_i = \sum_{j=0}^{M-1} k_{N-1+j-i} f_j, \quad i = 0, 1, \ldots, N-1, \tag{13}$$

where $u_i = u(x_i)$, $f_j = f(x_j)$, and $k_{N-1+j-i} = K(x_j - y_i)$. Discrete linear convolutions of this form can be casted into circular convolutions by appropriate padding of the vectors u and f. Performing these convolutions using DFTs leads to the fast FFT-based convolution technique given by

1. Pad sequence with zeros: append $N-1$ zeros to vector f.

$$\bar{f}_i = \big[\mathrm{Pad}(f)\big]_i = \begin{cases} f_i & i = 0, 1, \ldots, N-1 \\ 0 & i = N, N+1, \ldots, N+M-2 \end{cases} \tag{14}$$

2. Froward DFT: compute the DFT of sequences $\bar{f}$ and k via FFTs.

$$\hat{f} = \mathrm{FFT}(\bar{f}), \qquad \hat{k} = \mathrm{FFT}(k) \tag{15}$$

3. Convolution of DFTs: multiply complex coefficients of $\hat{f}$ and $\hat{k}$.

$$\hat{u}_i = \big[\mathrm{Prod}(g, k)\big]_i = \hat{f}_i \hat{k}_i, \quad i = 0, 1, \ldots, N+M-2 \tag{16}$$

4. Backward DFT: compute the inverse DFT of sequence $\hat{u}$ via FFT.

$$\bar{u} = \mathrm{FFT}^{-1}(\hat{u}) \tag{17}$$

5. Truncate sequence: remove the first $M-1$ entries of $\bar{u}$ to obtain u.

$$u_i = \big[\mathrm{Trunc}(\bar{u})\big]_i = \bar{u}_{M+i}, \quad i = 0, 1, \ldots, N-1 \tag{18}$$

This technique requires $\mathcal{O}((N+M)\log(N+M))$ operations and is readily generalized to higher dimensions for the case of tensor-product grids by recursively applying the 1D version to each directions.

2.3. Adaptive block-structured grid

Fast convolutions via FFTs discussed in Section 2.2 can be used to accelerate the evaluation of Eq. (5). In order to use this technique, the support of f needs to be padded with zeros to form a box. Similarly, the region where u is evaluated needs to be extended to also form a box. For cases where the domain defined by the support of the source terms is not a box, the cost of the additional computational elements can outweigh the reduced operation count per grid point of the FFT-based convolution technique.

Computational domains defined by the union of blocks can, however, be used to avoid excessive padding and still retain sufficient regularity to benefit from the fast FFT-based convolution technique. Our formulation partitions the infinite grid into blocks defined on a logically Cartesian grid. Blocks can potentially have a different number of grid points in each direction, but all blocks are required to have the same dimensions. An active source block denotes a block containing non-zero sources. Similarly, an active evaluation block denotes a block containing grid points on which the induced field is evaluated. The union of active source (evaluation) blocks is referred to as the active source (evaluation) grid. We emphasize that grid adaptivity is achieved through the selective choice of active blocks in order to define efficient computational domains, the present method does not consider problems with multiple spatial resolutions.

Let B_s and B_e denote the sets of active source and evaluation blocks, respectively. The convolution given in Eq. (5) can be evaluated by

$$u^P = \sum_{Q \in B_s} \mathrm{conv}\big(k^{Q-P}, f^Q\big), \quad \forall P \in B_e, \tag{19}$$

where u^P and f^P denote vectors containing the values of u($\mathbf{n}$) and f($\mathbf{n}$), respectively, evaluated on the grid points belonging to block P. Similarly, k^{Q-P} denotes the vector containing the unique values of $\mathsf{G}(\mathbf{m}-\mathbf{n})$, as described in Section 2.2, for values of $\mathbf{n}$ and $\mathbf{m}$ corresponding to the indices of grid points belonging to block Q and P, respectively. If blocks P and Q are sufficiently well-separated then $A^q_{\mathsf{G}}(\mathbf{m}-\mathbf{n})$ is used instead of $\mathsf{G}(\mathbf{m}-\mathbf{n})$ for constructing k^{Q-P}. The operator $\mathrm{conv}(k^{Q-P}, f^Q)$ denotes the generalization of Eq. (12) to arbitrary dimensions. Details regarding the construction of vectors u^P, f^P, and k^{Q-P} are omitted, since it immediately follows the discussion regarding the tensor-product grid generalization of Eq. (12).

Computing each instance of $\mathrm{conv}(k^{Q-P}, f^Q)$ in Eq. (19) using the fast FFT-based convolution technique leads to a scheme that evaluates Eq. (5), for the case of $B_s = B_e$, in $\mathcal{O}(N_B^2 N_b \log(N_b))$ operations, where N_B is the number of blocks belonging to B_s, and N_b is the number of grid points belonging to each block. The operation count can be further reduced to $\mathcal{O}(N_B N_b \log(N_b) + N_B^2 N_b)$ if the DFT of the kernel blocks, $\hat{k}^{Q-P}$, are pre-computed, and if the DFT of source and evaluation blocks are preformed as pre-processing and post-processing step, respectively. Details regarding pre-computations, and pre- and post-processing steps are discussed in subsequent sections.

2.4. Fast convolutions via interpolation-based kernel compression

Interpolation-based FMMs obtain a low-rank representation of the kernel, $K(\mathbf{x},\mathbf{y})$, by projecting it onto a finite basis of interpolation functions. Consider a function $f(\mathbf{x})$ sampled at n points, $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$. An approximation for $f(\mathbf{x})$ is given by

$$\tilde{f}^n(\mathbf{x}) = \sum_{i=0}^{n-1} \phi_i(\mathbf{x}) f(\mathbf{x}_i), \tag{20}$$

where $\phi_i(\mathbf{x})$ is a interpolation function associated with the interpolation node $\mathbf{x}_i$. An approximation for $K(\mathbf{x},\mathbf{y})$ is obtained by recursively applying Eq. (20) to each argument of $K(\mathbf{x},\mathbf{y})$,

$$\tilde{K}^n(\mathbf{x},\mathbf{y}) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \psi_i(\mathbf{x}) K(\mathbf{x}_i,\mathbf{y}_j)\phi_j(\mathbf{y}), \tag{21}$$

where $\{\phi\} = \{\phi_0, \phi_1, \ldots, \phi_{n-1}\}$ and $\{\psi\} = \{\psi_0, \psi_1, \ldots, \psi_{n-1}\}$ are potentially distinct bases of interpolation functions. In order to make the kernel-compression technique symmetric, only schemes with $\{\phi\} = \{\psi\}$ are considered. Directly applying this kernel compression technique to discrete convolutions of the form of Eq. (1), leads to an approximation of $u(\mathbf{x}_i)$ given by

$$u(\mathbf{x}_i) \approx \sum_{j=0}^{M-1}\sum_{p=0}^{n-1}\sum_{q=0}^{m-1} \phi_p(\mathbf{x}_i) K(\mathbf{x}_p,\mathbf{y}_q)\phi_q(\mathbf{y}_j) f(\mathbf{y}_j), \quad i = 0, 1, \ldots, N-1. \tag{22}$$

For cases involving multiple sets of either evaluation points, $\mathbf{x}_i$, or source points, $\mathbf{y}_j$, it is advantageous to decompose the evaluation of Eq. (22) into three steps:

1. Regularization: compute effective source terms using the adjoint of the interpolation procedure.

$$\tilde{f}(\mathbf{y}_q) = \sum_{j=0}^{M-1} \phi_q(\mathbf{y}_j) f(\mathbf{y}_j), \quad q = 0, 1, \ldots, n-1 \tag{23}$$

2. Convolution: compute the field induced by effective source terms on interpolation nodes.

$$\tilde{u}(\mathbf{x}_p) = \sum_{q=0}^{n-1} K(\mathbf{x}_p,\mathbf{y}_q)\tilde{f}(\mathbf{y}_q), \quad p = 0, 1, \ldots, n-1 \tag{24}$$

3. Interpolation: compute the field at evaluation points using the interpolation procedure.

$$u(\mathbf{y}_i) = \sum_{p=0}^{n-1} \phi_p(\mathbf{x}_i)\tilde{u}(\mathbf{x}_j), \quad i = 0, 1, \ldots, N-1 \tag{25}$$

If the values of $\phi_q(\mathbf{x}_i)$ and $\phi_q(\mathbf{y}_j)$ are known, the number of operations required by this procedure, for the case of $M = N$, is $\mathcal{O}(nN + n^2)$. For the case of $n << N$ this procedures represents a significant reduction in the number of operations compared to straightforward method of evaluating Eq. (1).

2.5. Fast convolution on regular grids using polynomial interpolation and FFTs

The fast convolution techniques presented in Sections 2.2 and 2.4 can be combined to yield a faster method for evaluating the block-wise convolutions involved in Eq. (19). This technique follows from the observation that Eq. (24) can be evaluated using FFTs if the kernel is translation-invariant and the interpolation nodes on which the kernel is evaluated are restricted to be on a regular grid. The first requirement is assumed since the kernels corresponding to the fundamental solutions of the linear, constant-coefficient elliptic difference equations are translation-invariant. The second condition is achieved by using an interpolation scheme based on equidistant interpolation nodes on tensor-product grids. Although many interpolation schemes satisfy the latter condition, only polynomial interpolation schemes on tensor-product grids are presently considered since they are fast, simple to implement, and their behavior is well-understood.[3]

Polynomial interpolation on tensor-product grids is performed by recursively applying 1D polynomial interpolation along each direction. This generalization has the advantage of maintaining the number of operation per grid point independent of dimension, and allows the behavior of the interpolation process to readily generalized from its 1D version.

In the absence of rounding errors, 1D polynomial interpolants converge geometrically if the function being interpolated is analytic in a region on the complex plane near the interpolation interval. The size and shape of this convergence region depends on the choice of interpolation nodes [27]. The kernels being considered correspond to the asymptotic expansion of the LGFs that are only discontinuous at the origin. Although convergences conditions should be verified for each kernel, the requirement that source and evaluation blocks be sufficiently well-separated to accurately evaluate the LGF using its asymptotic expansion is often sufficient to guarantee convergence.

Unlike Chebyshev interpolation, polynomial interpolation on equidistant nodes is ill-conditioned. Ill-conditioning can cause rounding errors due finite numeric precision to be amplified. The Lebesgue constant, $\Lambda_n(X)$, for a set of n interpolation points $X = x_0, x_1, \ldots, x_n - 1$, can be used to bound the growth of perturbations in the data [28],

$$\max_{x\in I}\left|p(x)-\tilde{p}(x)\right| \leq \Lambda_n(X)\max_{0\leq i\leq n-1}\left|f(x_i)-\tilde{f}(x_i)\right|, \tag{26}$$

where I is the interpolation interval, $p(x)$ and $\tilde{p}(x)$ are the polynomials interpolants resulting from the nodal values $f(x_i)$ and $\tilde{f}(x_i)$, respectively. The Lebesgue constant is given by

$$\Lambda(X) = \max_{x\in I}\sum_{i=0}^{n-1}\left|\phi_i(x)\right|, \tag{27}$$

where $\phi_i(x)$ is the Lagrange characteristic polynomial associate with x_i. Eq. (26) can be extended to polynomial interpolation on tensor product grids, with equal number of points and spacing in each direction, by replacing $\Lambda(X)$ with $(\Lambda_n(X))^d$, where d is the dimension of the problem.

In the limit of very large n, the Lebesgue constant of a set of equally spaced nodes is known to grow exponentially [28]. In order to avoid very large Lebesgue constants, the present scheme restricts number nodes used for polynomial interpolation to be at most $n_{\max}$. If $n_{\max}$ nodes are insufficient to achieve a desired interpolation error, additional nodes are added to the interval, but only the closest $n_{\max}$ nodes to the evaluation point are used for interpolation. Thus, this hybrid scheme performs both p- and h-refinement to increase the accuracy of the interpolations procedure. As a result, geometric convergence rates are expected for $n \leq n_{\max}$, and polynomial convergence rates of order $n_{\max} - 1$ are expected for $n > n_{\max}$. The values of n and $n_{\max}$ required to interpolate a function $f(x)$ over an interval I with an interpolation error less than ϵ are obtained in two steps:

1. Find the largest $n_{\max}$ such that $\Lambda_{n_{\max}}(X)\epsilon_p$ is less than ϵ, where ϵ_p is the precision of the floating-point scheme.
2. Progressively increase the number of n until the difference between $f(x)$ and it approximation are less than ϵ.

The same procedure can be used in higher dimensions by having n and $n_{\max}$ correspond to the number of interpolation points along each direction, and replacing $\Lambda_{n_{\max}}(X)$ with $(\Lambda_{n_{\max}}(X))^d$. For example, approximating an analytic function in 3D using double-precision arithmetic to relative tolerance of $\epsilon = 1.25 \times 10^{-12}$ requires $n_{\max} \leq 10$.

We omit a step-wise description of the combined fast algorithm for block-wise convolutions, since it readily follows from the discussion. Instead, we introduce the notation $\tilde{f}^P = \mathrm{Interp}(f^Q)$ and $f^Q = \mathrm{Reg}(\tilde{f}^P)$ corresponding to the interpolation and regularization (adjoint of interpolation) operations, respectively. Vector f^P contains the values of f($\mathbf{n}$) evaluated on the grid points of block P. Similarly, vector $\tilde{f}^Q$ contains the values of $\tilde{f}(\mathbf{x})$ evaluated on the grid points of the interpolation interval Q.

[3] Alternative interpolation procedures, with the exception of Fourier interpolation on non-periodic domains, have not been explored. Although Fourier interpolation is particularly appropriate given FFTs are used in our method to accelerate local computations, preliminary results have shown that this procedure is less efficient (in terms of points per unit accuracy) than the procedure described in this section.

3. The fast lattice Green's function method

3.1. Basic algorithm

Thus far we have discussed methods for accelerating the evaluation Eq. (19) by performing fast block-wise convolutions involving interpolation-based kernel compression and/or FFT techniques. Asymptotically these schemes require $\mathcal{O}(N^2)$ operations, though the constant in front of the N^2 term can be significantly smaller compared to that of the straightforward method. For kernels that decay or exhibit progressively smoother behavior away from the origin, for example the fundamental solution of the discrete Laplace operator, it is possible to combine the fast block-wise convolution techniques discussed in Sections 2 with the multilevel scheme of the original FMM [16]. To facilitate the discussion, we will assume that the active evaluation grid is the same as the active source grid.

Our multilevel scheme follows a tree (octree in 3D) structure similar to that described in [16]. Tree nodes at all levels are said to correspond to intervals.[4] Each interval is defined by the tensor-product grid associated with the nodes of the interpolation scheme. The tree is constructed by first creating one tree leaf, i.e. tree node with no children, for each active grid block. The intervals of tree leaves are defined such that they occupy the same spatial region as their associated grid blocks. After defining all tree leaves, sibling are recursively merged to generate the multilevel structure. We use the convention that all tree leaves are located at level 1 and that the root of the tree is located at level L.

The set of intervals at level ℓ is denoted by B^ℓ, and N_B^ℓ denotes the size of B^ℓ. In order to facilitate the discussion, intervals at level ℓ are chosen to contain n_b^ℓ nodes in each directions. The total number of points in each interval at level ℓ is given by $N_b^\ell = (n_b^\ell)^d$, where d is the dimension of the problem. The set of blocks defining the underlying active grid is denoted by B^0. N_B^0, n_b^0, and N_b^0 have analogous definitions to N_B^ℓ, n_b^ℓ, and N_b^ℓ, respectively. We note that level zero, $\ell = 0$, is not part of the tree structure, but the slight abuse of notation facilitates the description of the algorithm.

By construction, all intervals are Cartesian grids. As a result the union of intervals belonging to the same level defines an analogous grid to that of the underlying adaptive block-structured grid. Therefore, the techniques for fast block-wise convolutions discussed in Section 2 are readily generalized to all levels of the tree structure.

Our overall algorithm for solving systems of difference equations of the form given by Eq. (4) on adaptive block-structured grids is referred to as the Fast Lattice Green's Function (FLGF) method. The FLGF method is described by following steps:

0. *Pre-computation*: compute and store all unique $\hat{k}^{P-Q}$ used in Step 2.

$$\hat{k}^P = \mathrm{FFT}\big(k^{P-Q}\big) \tag{28}$$

1. *Upwards Pass*: $\forall P \in B^\ell$, for $\ell = 0, 1, \ldots, L$
   (a) Regularize: compute effective source terms at interpolation nodes

$$\tilde{f}^P = \sum_{Q \in \mathrm{RegSupp}(P)} \mathrm{Reg}\big(\tilde{f}^Q\big), \tag{29}$$

   (b) Padded forward DFT: prepare vectors for DFT convolutions

$$\hat{f}^P = \mathrm{FFT}\big(\mathrm{Pad}\big(\tilde{f}^P\big)\big), \tag{30}$$

2. *Level Interactions*: $\forall P \in B^\ell$, for $\ell = 0, 1, \ldots, L$

$$\hat{u}^P = \sum_{Q \in \mathrm{InfluenceList}(P)} \mathrm{Prod}\big(\hat{f}^P, \hat{k}^{P-Q}\big), \tag{31}$$

3. *Downwards Pass*: $\forall P \in B^\ell$, for $\ell = L, L-1, \ldots, 0$
   (a) Truncated backwards DFT: extract relevant data from DFT convolution

$$\tilde{v}^P = \mathrm{Trunc}\big(\mathrm{FFT}^{-1}\big(\hat{u}^P\big)\big) \tag{32}$$

   (b) Interpolate: compute and aggregate the induced field at interpolation nodes

$$\tilde{u}^P = \tilde{v}^P + \mathrm{Interp}\big(\tilde{u}^{\mathrm{InterpSupp}(P)}\big) \tag{33}$$

We note that the operations performed in Steps 1, 2, and 3 are commonly referred to as the multipole-to-multipole, multipole-to-local, and local-to-local operations, respectively, in the FMM literature. The lists of blocks/intervals used by the algorithm are given by

[4] In the context of the hierarchical algorithm and structure, the term "interval" is equivalent to term "box" used in [16]. We reserve the term "box" for geometric descriptions, and do not associate any specific structure or information with the term.

Table 1
Cost per interval/block of present FMM method.

	Cost	Order
Pre-computations	$C^{\ell}_{\text{EvalKernel}} + C^{\ell}_{\text{PadFFT}}$	$N_b^{\ell} \log N_b^{\ell}$
Upwards pass ($\ell = 0$)	C^{0}_{PadFFT}	$N_b^{0} \log N_b^{0}$
Upwards pass ($\ell = 1$)	$C^{\ell}_{\text{Interp}} + C^{\ell}_{\text{PadFFT}}$	$N_b^{\ell} \log N_b^{\ell}$
Upwards pass ($\ell > 1$)	$8C^{\ell}_{\text{Interp}} + C^{\ell}_{\text{PadFFT}}$	$N_b^{\ell} \log N_b^{\ell}$
Level interactions ($\ell = 0$)	$27C^{\ell}_{\text{Prod}}$	$27N_b^{0}$
Level interactions ($\ell > 0$)	$189C^{\ell}_{\text{Prod}}$	$189N_b^{\ell}$
Downwards pass	$C^{\ell}_{\text{Interp}} + C^{\ell}_{\text{PadFFT}}$	$N_b^{\ell} \log N_b^{\ell}$

$$\text{RegSupp}(P) = \begin{cases} \text{block } P & \text{if } P \in B^0 \\ \text{block associated with interval } P & \text{if } P \in B^1 \\ \text{children of interval } P & \text{if } P \in B^{\ell},\ 2 \le \ell \le L \end{cases} \tag{34}$$

$$\text{InterpSupp}(P) = \begin{cases} \text{interval associated with block } P & \text{if } P \in B^0 \\ \text{parent of interval } P & \text{if } P \in B^{\ell},\ 1 \le \ell < L \\ \emptyset & \text{if } P \in B^L \end{cases} \tag{35}$$

$$\text{InfluenceList}(P) = \begin{cases} \text{near-neighbors of block } P & \text{if } P \in B^0 \\ \text{interaction list of interval} & \text{if } P \in B^{\ell},\ 1 \le \ell \le L \end{cases} \tag{36}$$

Children, parents, near-neighbors, and interaction lists follow the same definitions those of the original FMM [16]. Based on these definitions, we note that Eq. (29) reduces to $\tilde{f}^P = f^P$ for $P \in B^0$, and Eq. (33) reduces to $\tilde{u}^P = \tilde{v}^P$ for $P \in B^L$.

3.2. Algorithmic complexity

The overall complexity of our algorithm is $\mathcal{O}(N)$, as is the case for the original FMM. For simplicity, the discussion concerning the cost of each step is limited to the 3D version of the algorithm. Details regarding to the cost of each block/interval are presented in Table 1. The factor of 8 in front of C^{ℓ}_{Interp} for the *Upwards Pass* ($\ell > 1$) is due to the fact that each interval has 8 children. The constants, 27 and 189, associated with the cost of *Level Interactions* correspond to the number of near-neighbors and the number of members of each interaction list, respectively.

The specific values and a brief discussion of the constants presented in Table 1 is provided:

1. $C^{\ell}_{\text{EvalKernel}}$: Cost of kernel evaluation performed in Eq. (28). Constructing the vector k^{Q-P}, where Q and P are blocks/intervals at level ℓ, requires $8N_b^{\ell}$ kernel evaluations. For small values of $|\mathbf{n} - \mathbf{m}|$ a look-up table is used to evaluate $\mathsf{G}(\mathbf{n} - \mathbf{m})$, otherwise there kernel is evaluated using $A^n_{\mathsf{G}}(\mathbf{n} - \mathbf{m})$.
2. C^{ℓ}_{Interp}: Cost of polynomial interpolation performed in Eq. (33). The coefficient mapping interpolation nodes to evaluation nodes are precomputed (only needed for 1D interpolation); therefore computing the values of a single block/interval at level ℓ requires $\min(n_b^{\ell+1}, n_{\max})N_b^{\ell}$ operations (1 real addition and 1 real multiplication per operation), where $n_{\max}$ is described in Section 2.5. In 3D, $n_{\max}$ is typically set to be no greater than 10. C^{ℓ}_{Interp} also describes the cost of performing the regularization, adjoint of interpolation, operation involved in each term of the sum in Eq. (29).
3. C^{ℓ}_{PadFFT}: Cost of performing a 3D FFT (real-to-complex) or inverse FFT (complex-to-real) on the padded vectors present in Eqs. (28), (30), and (32). The operation count (total number of real additions and multiplications) for each FFT performed using the FFTW library is approximately $2(8N_b^{\ell}) \log_2(8N_b^{\ell})$ [29], where $8N_b^{\ell}$ is the size of the padded vectors. Since all FFTs are real-to-complex or complex-to-real approximately half of the coefficients are redundant and need neither be stored nor operated on.
4. C^{ℓ}_{Prod}: Cost of performing DFT convolutions, i.e. multiplication of complex coefficients, in Eq. (31). If blocks/intervals Q and P belong to level ℓ, then performing $\text{Prod}(\hat{f}^P, \hat{k}^{P-Q})$ requires approximately $4N_b^{\ell}$ operations (1 complex multiplication per operation), where $4N_b^{\ell}$ is the number of non-redundant DFT coefficients per block/interval.

The cost per level of each operation, except for the *Pre-computation* step, can be obtained by multiplying the values of each row by the N_B^{ℓ}. In regards to the *Pre-computation* step, the cost per level for $\ell = 0$ and $\ell > 0$ is obtained by multiplying the cost per block/interval by 27 and 317, respectively, which correspond to the number of unique $\hat{k}^{P-Q}$ vectors that are used at each level. If the kernel shares the same symmetry as the LGF of the discrete Laplace operator, the number of unique $\hat{k}^{P-Q}$ vectors per level is reduced to 4 and 36 for $\ell = 0$ and $\ell > 0$, respectively. If symmetry is used to reduce number of pre-computed $\hat{k}^{P-Q}$ vectors, then C^{ℓ}_{Prod} is roughly doubled since a twiddle factor needs to be applied to the DFT coefficients for cases involving reflections.

3.3. Parallel implementation

A brief overview of our MPI-based algorithm is included to demonstrate that the present method allows for a simple parallel implementation suitable for practical large-scale scientific computing. Our current implementation builds the tree structure and performs load balancing operations on a single MPI-process, and then scatters the information to all other process. As a result, all the information necessary to evaluate RegSupp, InterpSupp, and InfluenceSupp for any block/interval is known by all processes. The tree structure is constructed following the bottom-up approach discussed in Section 3.1. Prior to partitioning the problem, the load balancing scheme first assigns a weight to every block and interval based on an estimate of its runtime cost. Next, parent tree nodes are recursively grouped with one of its child tree nodes, and tree leaves are grouped with their associated grid blocks. The set of groups is then partitioned into clusters in such away that weight of each cluster (aggregate weight of all interval/blocks belonging to all groups in the cluster) is roughly the same. Each cluster is then assigned to an MPI process. Given each group contains only one block, a Morton or Z-order curve [30] is used to preserve data locality during partitioning.

The parallel algorithm closely resembles the serial algorithm, since each process executes steps analogous to the *Upwards Pass*, *Level Interactions*, and *Downwards Pass*. Non-blocking routines are used for all communications, allowing for computations to be overlapped with communications. Furthermore, our algorithms uses these routines to avoid any global synchronization within each steps and between steps.

The parallel execution of each step follows a similar event-driven paradigm, where processes performs "work units" based on the information they currently have and send information to other processes, or to itself, when sufficient "work units" have been completed. Send and receive buffers are used to avoid excessive memory requirements. Our algorithm gives priority to "work units" yielding results that are sent to other processes. The time spent waiting to either receive information or to clear send buffers is used to perform local "work units", i.e. operations that only require data and yield results pertinent to the same process. During the *Upward* and *Downward Pass* a non-blocking send is posted after each interval-/block-wise regulation and interpolation operation is completed. In contrast, during *Level Interactions* step the influence of all intervals/blocks belonging to a process on all intervals/blocks belonging another process is aggregated and packages before sending; thus each process sends at most one message to every other process during this step. For convenience, pseudo-codes for the communication patterns described in this section are given in Appendix B.

4. Numerical results

In this section we present numerical results that demonstrate the accuracy, computational cost, and parallel performance of the FLGF method. The results reported are for the discrete Poisson equation in 3D. The accuracy of the method is determined by the difference between the computed solution and u of the difference equation in Eq. (4), as opposed to u of the PDE in Eq. (2).

As in Section 3, the active evaluation grid is defined to be equal to the active source grid. For all test cases, blocks/intervals belonging to level ℓ are chosen to contain n_b^ℓ points in each directions. In the interest of brevity, we only consider schemes where the number of interpolation nodes per interval is same for all levels, $n_b^\ell = n_I$ for $\ell = 1, 2, \ldots, L$, and require that $n_I = n_b^0 + 1$.[5] These considerations reduce the parameter space of possible schemes to cases where spacing between nodes of parent intervals is twice that of child intervals. Furthermore, there is effectively no regularization/interpolation between level 0 and level 1, since the nodes on level 1 coincide with the underlying grid points. Following the discussion of Section 2.5 we set maximum number nodes used for interpolating a value at single point, $n_{\max}$, to be 10 for all test cases.

We note that the previous restrictions are only imposed for the purpose of a concise exposition. The non-scale-invariant behavior of most LGFs suggests that, in general, non-trivial performance gains can be achieved by tuning n_b^ℓ for each level. Given the LGF of the discrete Laplace operator is approximately scale-invariant away from the origin, possible performance gains achieved by varying n_b^ℓ are not considered in the present discussion.

Our code is written in Fortran, and uses the third-party libraries MVAPICH2 and FFTW 3.3 to perform MPI-based commutations and to evaluate DFTs via FFTs, respectively. Numerical experiments are performed using our local computing facility which consists of 60 compute nodes connected by a QDR InfiniBand network. Each node containing 2 Intel Xeon X5650 processor (6-core, 12 MB cache, 2.66 GHz clock speed) and 48 GB of RAM.

4.1. Error

The accuracy of the proposed methodology is investigated on cubic active grids containing different numbers of active grid points and partitioned into blocks of different sizes. A procedure based on random manufactured solutions is used to determine the error of each test case. In this procedure a solution, $\mathrm{u}_{\mathrm{rand}}(\mathbf{n})$, is manufactured by assigning a random value between -1 and $+1$ to each grid points, except for grid points on the boundary of the active grid which are set to zero. The source distribution, $\mathrm{f}(\mathbf{n})$, which serves as the input for each test case, is computed by taking the discrete Laplacian of

[5] Our implementation requires for intervals belonging to $\ell > 0$ have a one grid point overlap with its neighboring intervals along (n_1, n_2, n_3) directions, where $n_i = \{0, 1\}$ and at least one n_i is non-zero.

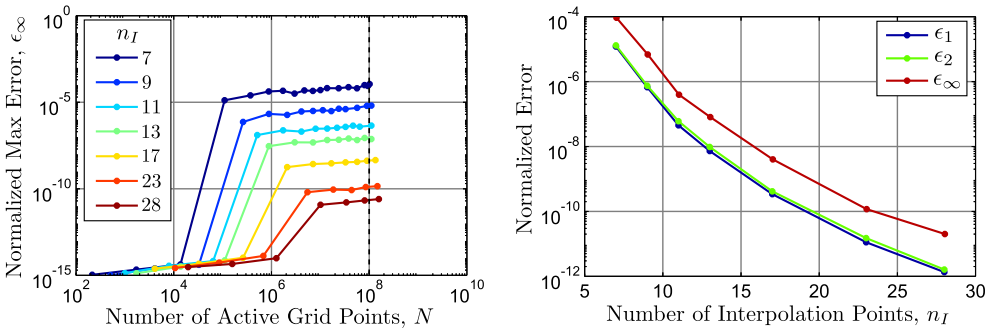

Fig. 1. *Left*: max error, ϵ_∞, for cubic active grids containing N grid points partitioned into blocks with $n_b^0 = n_I - 1$ grid points along each direction. *Right*: error for test cases containing 10^8 grid points as function of n_I; the curve corresponding to ϵ_∞ is equivalent to the values on the *left* plot intersecting the vertical dashed line.

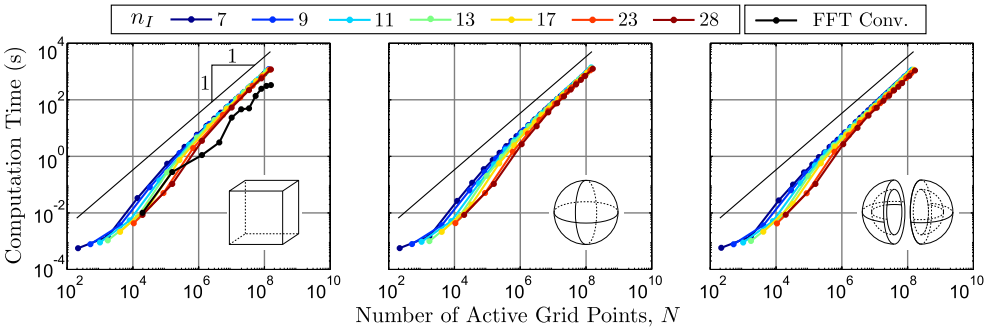

Fig. 2. Computation times for active grids containing N grid points partitioned into blocks with $n_b^0 = n_I - 1$ point in each direction. The curve labeled "FFT Conv." corresponds to the special cases where the entire active grid is a single block. Results are presented for active grids with geometries approximating: cubes (*left*), spheres (*middle*), and spherical-shells (*right*).

the prescribed solution, i.e. $\mathsf{f}(\mathbf{n}) = [\mathsf{L}\mathsf{u}_{\text{rand}}](\mathbf{n})$. The error for each test case is computed by $\epsilon_p = \|\mathsf{u} - \mathsf{u}_{\text{rand}}\|_p / \|\mathsf{u}_p\|_p$, where $\mathsf{u}(\mathbf{n})$ is determined by solving $[\mathsf{L}\mathsf{u}_p](\mathbf{n}) = \mathsf{f}(\mathbf{n})$, and $\|\mathsf{u}\|_p$ is the L^p-norm of u computed over the active grid. The error for various problem sizes and schemes based on different blocks sizes is presented in Fig. 1.

Test cases involving a small number blocks do not make use of the interpolation-based kernel compression technique; they only make use of the block-wise FFT-based convolution technique, which incurs an error close to machine precision. As a result, the three cases with the smallest values of N for each series have significantly smaller errors compared to their respective asymptotic (large N) error.

Given that we chose $n_{\max} = 10$ and constrained n_b^0 to be proportional n_I, the error is expected to decay as n_I^{-10} for $n_I > n_{\max}$ (one order greater than interpolation order due to the $\sim |\mathbf{x}|^{-1}$ decay of the LGF). The data shown Fig. 1 are consistent with our estimates, exhibiting a behavior proportional to $n_I^{-10.7}$ for values of n_I between 11 and 28. The significant difference in the magnitude of ϵ_1 and ϵ_2 compared to ϵ_∞ suggests that the maximum error is concentrated in lower dimensional regions of the grid. Spatial plots of the error (not included) confirm that larger errors are always observed near or on the boundary points of blocks. These observations are characteristic of the interpolation scheme being used, which is known to exhibit larger errors near the boundaries of the interpolation interval (Runge phenomena).

Although the error for schemes with $n_I \neq n_b^0 + 1$ are not presented, it is readily deduced from our reported results that for any choice of n_b^0 the error can be controlled by changing n_I. Furthermore, different choices of $n_{\max}$ where not explored since $n_{\max} = 10$ allows for schemes with errors as small as $\sim 10^{-12}$, which are sufficient for many practical applications. Errors smaller than 10^{-12} can be obtained by reducing $n_{\max}$ and increasing n_I.

4.2. Computation time

The test cases used to examine the computation time of the FLGF method follow the same setup as in Section 4.1, except that we now consider three types of active grid geometries: cubes, spheres, and spherical-shells (with a thickness of 0.1 diameters). For cases of spheres and spherical-shells, the union of blocks that define the active grid are distributed in such that way that they best approximate these geometries. Computation times for various schemes and problem sizes are presented in Fig. 2, and asymptotic computation rates (total number of active grid points/computational time) for a few schemes are included in Table 2.

As expected, the results for all schemes presented in Fig. 2, with the exception of the one labeled "FFT Conv.", have an asymptotic computational complexity of $\mathcal{O}(N)$. FFT Conv. refers to the special case where the entire active grid is a

Table 2
Asymptotic computation rates for selected test cases presented in Fig. 2. Values given in units of 10^5 pts/s. Rates are based on the test cases with 10^8 active grid points (values interpolated from nearest two data points). Asterisk ($*$) indicates rates that are not strictly asymptotic since they correspond to $\mathcal{O}(N \log N)$ schemes.

Scheme	Box	Sphere	Spherical-shell
$n_I = 7$	1.187	1.167	1.267
$n_I = 13$	1.189	1.169	1.315
$n_I = 28$	1.384	1.341	1.150
FFT Conv.	3.540*	n/a	n/a

single block for which the FLGF method reduces to a single FFT-based convolution. We note that in 3D the operation count of an FFT Conv. is roughly 8 times the operation count of solving the systems of equations obtained from the spectral discretization of differential operators on periodic domains using FFTs (referred to as "FFT Periodic" in the subsequent discussion). FFT Conv. is a useful point of comparison since (1) many methods can readily consider the case of a single block with uniformly distributed sources, (2) it is well-established that for regular grids FFT-based elliptic solvers achieve very high, if not the highest, computation rates, and (3) the performance of an algorithm relative to FFT Conv. is, approximately, hardware independent.

Fig. 2 demonstrates that the computation rates of our multi-block algorithm are within a factor of 10 of those corresponding to FFT Conv.. For large problems (our interest here), e.g. $N = \mathcal{O}(10^8)$, Table 2 indicates that our algorithm achieves computation rates that are roughly a third of the rate of FFT Conv. with up to 10 digits of accuracy. By comparison, the next closest method to ours, the 2D LGF FMM of Gillman and Martinsson [7], is claimed to be two orders of magnitude slower than an FFT Periodic. Even after accounting for the fact that in 2D an FFT Conv. is a factor of 4 slower than an FFT Periodic, we observe that our method has a significantly higher computation rate.[6] We also compare the performance of the present method to that of the black-box FMM of Fong and Darve [21]. In solving a 3D free-space Poisson problem with 10^6 uniformly distributed point sources over a cube with strengths ± 1, [21] reported a computation rate of about 1.8×10^3 points per second for 8 digits of accuracy.[7] Our method achieves a rate of about 1.2×10^5 points per second for 10 digits of accuracy, a speedup of about 65 times even with 2 more digits of accuracy. Finally, we observe that our computation rates are comparable with those of the adaptive (locally-refined) volume FMM of Langston et al. [31]. In solving a 3D free-space Poisson problem in a cube with sources corresponding to Gaussian bump solution, [31] reported computation rates between 2.6×10^4 and 1.3×10^5 points per second for cases with either 7 or 8 digits of accuracy (with number of unknowns between 2.8×10^6 and 2.6×10^7).[8]

A detail report of the computation time of the *Pre-computation* step is omitted, since this step is observed to require only a small fraction of the computation time of a single solve. We substantiate this claim by reporting that for test cases involving cubic active grids containing $\mathcal{O}(10^1)$, $\mathcal{O}(10^2)$, and $\mathcal{O}(10^4)$ blocks the *Pre-computation* step required less than 10%, 1%, and 0.1%, respectively, of the computation time of a single solve. These results are consistent with the fact that the operation count of the pre-computation step increase with the number of levels, which in turn increase logarithmically with the number of blocks for test cases involving cubic active grids.

4.3. Parallel performance

The parallel performance of the FLGF method is investigated by considering cubic active grids and using the scheme corresponding to $n_I = 13$ described in Sections 4.1 and 4.2. Computation rates and parallel efficiencies for various problem sizes with core counts between 12 and 660 are included in Fig. 3. For all reported test cases the number of cores is a multiple of 12 (there are 12 cores per node in our test machine), and each MPI-process is mapped to a single core. The parallel efficiency for each test series is defined by

$$e_N(p) = \frac{p_{\min}}{p} \frac{T_N(p_{\min})}{T_N(p)}, \tag{37}$$

where N is the total number of active grid points in the test series, p is the number of cores, $p_{\min}$ is the minimal number of cores considered in the test series, and $T_N(p)$ is the runtime of the test problem.

[6] For the numerical experiments reported, [7] stated that "the method was run at a requested relative precision of 10^{-10}". Given that [7] did not report the error of the experiments, we assume that computation rates quoted are for approximately 10 digits of accuracy, which is the same accuracy of the rates reported for the present method.

[7] We note that 8 digits is the maximum accuracy [21] reported for the 3D Laplace kernel. Additionally, we note that the hardware [21] used to perform the numerical experiments is not reported; therefore no attempt has been made to account for hardware differences.

[8] The numerical experiments reported in [31] were performed using a shared-memory (OpenMP) implementation running on 16 cores of an Intel Xeon X7560 (2.27 GHz) based system. The rates included in the main text correspond to the rates we would expect to observe if the numerical experiments of [31] were performed on a single core of our local system; both the parallel efficiency (reported to be approximately 75% for 16 cores) and the difference in processor clock speed have been accounted for.

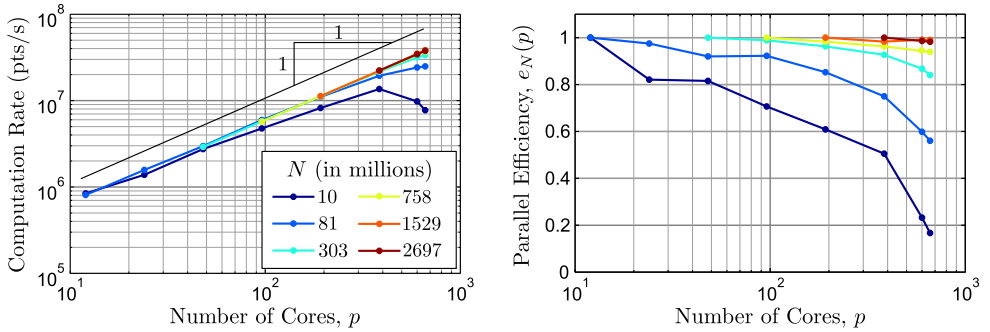

Fig. 3. Computation rates (*left*) and parallel efficiencies (*right*) for cubic active grid of various sizes. The parameter n_I is set to 13 for all test cases. The listed values of N, in ascending order, correspond to problems containing 5.8×10^3, 4.6×10^4, 1.8×10^5, 4.4×10^5, 8.3×10^5, and 1.6×10^6 active blocks.

Both strong and weak scaling can be inferred from the left plot in Fig. 3. Strong scaling, i.e. fixed N and increasing p, corresponds to the individual curves associated with each test series. Weak scaling, i.e. fixed N/p and increasing p, is achieved when the curves associated with different test series collapse. Fig. 3 demonstrates that, over a reasonable range p, the curves for most of the test series collapse to a single line of approximately unit slope. This indicates that our implementation exhibits both good strong and weak scaling.

There are two main considerations that affect the performance of our parallel implementation. The first is the number of blocks per core. The FLGF method is broken into block-wise operations. If there too few blocks per core our total work cannot be evenly distributed across all cores. Furthermore, given our communication scheme for the *Level Interactions* step, fewer blocks per core is likely to increase the total number MPI messages sent and received. The second consideration is the amount work per core. If the work per core is too small then communication cost can take an overwhelming fraction of the net runtime. Based on these considerations and the reported results, we conclude that if each core has, on average, more than 300,000 active grid points and 200 blocks then the parallel efficiency, as defined in Eq. (37), is expected to be above 80%. This observation seems consistent with the results reported on other MPI-based implementations of kernel-independent FMMs, for example [32,33].

In the interest of completeness, we note that computation rates reported in Fig. 3, in particular those corresponding to 12 cores, are roughly half the rates expected based on the our serial results. This decrease in performance due to an increase in cache-misses when more than one core per node is used. We expect that future, higher-quality, implementations of the FLGF method can readily mitigate this feature.

5. Conclusions

We have presented a new kernel-independent fast multipole method for elliptic difference equations on infinite Cartesian grids. The FLGF method exploits the regularity of the underlying grid to achieve small computation times by using a fast convolution technique that combines interpolation-based kernel compression and FFTs. Interpolation based on equidistant nodes, along with a p- and h-refinement technique, is shown to be an effective scheme for obtaining low-rank representations of kernels, while still preserving sufficient regularity to allow discrete convolutions to be performed quickly using FFTs. The adaptive block-structured grid strategy blends well with the overall algorithm, and the reported numerical experiments demonstrate that computation rates remain roughly invariant of source distributions.

The efficiency of the FLGF method is demonstrated through several numerical experiments solving the discrete 3D Poisson equation for cases involving up to 2 billion grid points and 660 cores. Serial test cases confirm that the algorithm archives an asymptotic linear complexity. Computation rates of approximately 1.2×10^5 pts/s or, equivalently, grind times of 8.3 μs are observed for problems containing 10^8 grid points. The computation rate is shown to be roughly invariant to different source distributions and block sizes. Furthermore, the time required to perform all pre-computations for typical problems is shown to negligible. Test cases investigating the parallel performance of our implementation demonstrate that parallel efficiencies higher than 80% are achieved under modest considerations (at least 200 blocks and 300 000 grid points per core).

The FLGF method is particularly useful for solving PDEs that have been discretized using a numerical scheme that enforces discrete conservation laws. In such cases, accurate solutions to the difference equations, but not necessarily the original PDE, are necessary to preserve physical fidelity. We have applied the present method to solve incompressible, viscous, external flows using a finite volume scheme and an infinite staggered Cartesian grid. Fig. 4 includes a snapshot of thin vortex ring at a Reynolds number (based on ring circulation) of 7500 simulated using this scheme. A detailed description and results pertaining to the application of the FLGF method to the incompressible Navier–Stokes is the subject of future publications.

In the interest of brevity, our discussion and reported results only pertain to the discrete Laplace kernel, yet the FLGF method can be applied to other non-oscillatory LGFs. In fact, our method can be readily generalized to any non-oscillatory kernel (including singular kernels); the only restriction is that sources and evaluation points be defined on a regular grid.

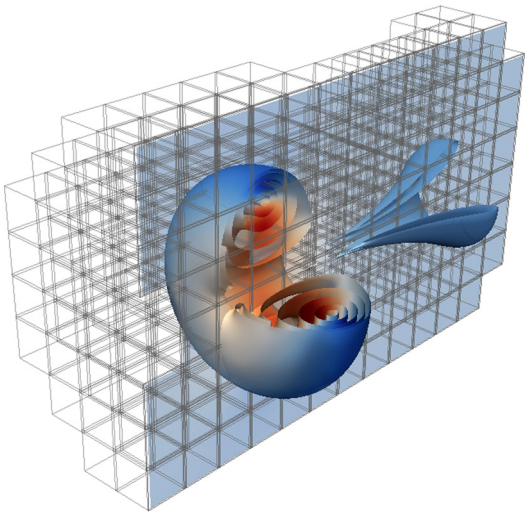

Fig. 4. Vortex ring at a Reynolds number of 7500. Isocontours correspond to the absolute value of vorticity (log scale), color corresponds to the streamwise velocity, and gray boxes correspond to the location of grid blocks used in the simulation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Based on these observations, and the simple/standard routines and data-structures involved in the algorithm, it is expected that the FLGF method can be readily incorporated into a wide range of existing methods and codes that solve elliptic PDEs on unbounded domains.

Appendix A. Numerical evaluation of the fundamental solution of the Laplace operator

Values of $\mathsf{G}(\mathbf{n})$ for small $|\mathbf{n}|$ are frequently used by the FLGF method. Therefore it is advantageous to program an accurate look-up table for values of $\mathbf{n}$ confined to a small cubic box centered at the origin. The symmetry of $\mathsf{G}(\mathbf{n})$, discussed in Section 2.1, suggests that only approximately 1/48-th of the total number of points in the box need to be numerically evaluated. It is possible to reduce the triple integral in Eq. (6) to a single semi-infinite integral [34] given by

$$\mathsf{G}(\mathbf{n}) = -\int_0^\infty e^{-6t} I_{n_1}(2t) I_{n_2}(2t) I_{n_3}(2t)\, dt, \tag{A.1}$$

where $I_k(x)$ is the modified Bessel function of the first kind of order k, and $\mathbf{n} = (n_1, n_2, n_3) \in \mathbb{Z}^3$. In our experience, it is easier and faster to numerically integrate Eq. (A.1) instead of Eq. (6). The integrand of Eq. (A.1) is non-oscillatory and smooth throughout the domain of integration. A simple adaptive Gauss–Kronrod scheme can be used to perform the numerical integration and obtain error estimates. Furthermore, the semi-infinite integral can be partitioned into two intervals $[0, \alpha]$ and $[\alpha, \infty]$, where α is chosen such that the latter integral can be evaluated analytically using the asymptotic expansion (for large arguments) of $I_n(x)$ [35]. More efficient implementations might considered partitioning the integration interval into multiple subintervals, exploiting both the ascending series representation and the asymptotic expansion of each Bessel function.

Appendix B. Communication patterns of MPI-based implementation

The pseudo-codes provided in this appendix complement the discussion regarding the parallel implementation of the present method included in Section 3.3. For convenience, in this appendix the term "node" is used to denote either grid blocks, grid intervals, or tree-nodes as defined in Section 3.1 and its precise meaning is deduced from the context. The algorithms discussed in this appendix are based on non-blocking MPI operations; we refer the reader to [36] for an introduction to these operations.

B.1. Level Interactions

A pseudo-code for the parallel implementation of *Level Interactions*, Step 2 of the FLGF algorithm of Section 3.1, is provided in Algorithm 1.

Algorithm 1: Communication pattern of *Level Interactions*.

```
while not done sending or receiving or local-work do
    check status of all active messages;
    forall the receive-messages that have completed receive do
        process receive-message;
        mark receive-buffer unit associated with receive-message as available;
    forall the available receive-buffer units do
        if not done posting receive-messages then
            associate receive-message with receive-buffer unit;
            post non-blocking receive for receive-message;
    forall the send-messages that have completed send do
        mark send-buffer associated with send-message as available;
    if not done with all send-work units and send-buffer unit is available then
        if send-work unit corresponds to new send-message then
            associate send-message with send-buffer unit;
        perform M send-work units of send-message;
        if done building send-massage then
            post non-blocking send for send-message;
    else if not done with all local-work units then
        perform N local-work units;
```

A description of the terms and operations used by Algorithm 1 is provided below.

Done sending (receiving): all non-blocking MPI messages being sent (received) have been posted and completed.
Done local-work: any intra-MPI-process operations have been completed.
Send-/receive-messages: As described in Section 3.3, each MPI-process sends, at most, one message to any other MPI-process. A message is composed of sub-messages; one for each target node. Each sub-message contains information identifying the target node, and the field induced on target node by all source nodes that belong to the sending MPI-process and that interact with target node.
Buffer and buffer units: messages are buffered before being posted. The send-buffer and receive-buffer are composed of a fixed number of buffer-units. Each buffer-unit is allocated enough memory to handle any message that will be posted. The examples included in Section 4.3 use a total of 4 buffer units, 2 for sending and 2 for receiving.
Process receive-message: read receive-message and add field contribution from non-local source nodes to local target nodes.
Send-work unit: compute the field induced from a single local node to a target node belonging to the current target MPI-process. The induced field of each target node is aggregated in send-buffer. Operations performed by a single send-work unit correspond to those of a single entry of the sum given in Eq. (31).
Local-work unit: same as send-work unit, except that the induced field is added to the local storage of the target node (no need to buffer or perform any MPI communication).
Done building send-message: the results from all send-work units required by send-message (or, equivalently, target MPI-process) have been packaged into a message.
Parameters M and N: determine the number of work units performed at each iteration of the main loop. The examples included in Section 4.3 use $M = N = 20$.

B.2. Upwards and downwards pass

A pseudo-code for the parallel implementation of *Upwards Pass*, Step 1 of the FLGF algorithm of Section 3.1, is provided in Algorithm 2.
A description of the terms and operations used by Algorithm 2 is provided below.

Done sending (receiving) and done local-work: same as in B.1.
Message: a messages contain a node's weights (sources or regularized weights/sources from child nodes). Each node is provided enough auxiliary storage to receive the weights from all of its children.
Receive-tracker: a tree-like data-structure that contains information regarding the progress of all communication and operations associated with each local node. It can be used to determine whether a node has finished receiving messages from all of its children (if any), whether a padded-FFT has been performed on its weights, and whether it has posted a non-blocking send to its parent (if any).
Ready-for-processing node: a node that has not be processed, but has finished receiving messages from all of its children (if any). A node is said to be processed if its weights have been computed (or are known), i.e. Eq. (29), a padded-FFT

Algorithm 2: Communication pattern of *Upwards Pass*.

```
forall the local nodes do
    forall the non-local children of node do
        post non-blocking receive for message sent by child;
while not done sending or receiving or local-work do
    check status of all active messages and update receive-tracker;
    select M ready-for-processing nodes using receive-tracker;
    forall the selected nodes do
        if node has children then
            build node's weights by regularizing child nodes' weights;
        perform padded-FFT on node's weights;
        if node has a parent then
            if parent is non-local then
                post non-blocking send for message received by parent;
            else
                update local parent node's weights;
```

has been performed on its weights, i.e. Eq. (30), and a non-blocking send to its parent-node has been posted (if parent exists).

Selecting ready-for-processing nodes: the receive-tracker is transversed in leaf-to-root order and nodes that meet the ready-for-processing criteria are selected. Priority is given to nodes whose parent are non-local, i.e. belong to a different MPI-process.

Parameter M: determine the number of nodes to be processed at each iteration of the main loop.

Pseudo-code for *Downwards Pass* is omitted since the communication pattern is very similar that of *Upwards Pass*. The only significant differences are that nodes send to their children, as opposed to their parent, and that when selecting ready-for-processing nodes the receive-tracker is transversed in root-to-leaf order, as opposed to leaf-to-root order. The operations preformed at each step and the ready-for-processing criteria are readily deduced from the discussion included in Sections 3.1 and 3.3.

References

[1] W.H. McCrea, F.J.W. Whipple, Random paths in two and three dimensions, Proc. R. Soc. Edinb. 60 (1940) 281–298.
[2] E.W. Montroll, R.B. Potts, Effect of defects on lattice vibrations, Phys. Rev. 100 (1955) 525–543.
[3] E.N. Economou, Green's Functions in Quantum Physics, Springer-Verlag, Berlin, 1984.
[4] J.H. Bramble, B.E. Hubbard, On the formulation of finite difference analogues of the Dirichlet problem for Poisson's equation, Numer. Math. 4 (1962) 313–327.
[5] O. Buneman, Analytic inversion of the five-point Poisson operator, J. Comput. Phys. 8 (1971) 500–505.
[6] A. Gillman, P.G. Martinsson, Fast and accurate numerical methods for solving elliptic difference equations defined on lattices, J. Comput. Phys. 229 (2010) 9026–9041.
[7] A. Gillman, P.G. Martinsson, A fast solver for Poisson problems on infinite regular lattices, J. Comput. Appl. Math. 258 (2014) 42–56.
[8] D.N. Arnold, P.B. Bochev, R.B. Lehoucq, R.A. Nicolaides, M. Shashkov, Compatible Spatial Discretizations, The IMA Volumes in Mathematics and its Applications, vol. 142, Springer New York, New York, 2006.
[9] J.B. Perot, Discrete conservation properties of unstructured mesh schemes, Annu. Rev. Fluid Mech. 43 (2011) 299–318.
[10] M.L. Glasser, I.J. Zucker, Extended Watson integrals for the cubic lattices, Proc. Natl. Acad. Sci. USA 74 (1977) 1800–1801.
[11] R.T. Delves, G.S. Joyce, On the Green function for the anisotropic simple cubic lattice, Ann. Phys. 291 (2001) 71–133.
[12] R.J. Duffin, E.P. Shelly, Difference equations of polyharmonic type, Duke Math. J. 25 (1958) 209–238.
[13] M. Mangad, Asymptotic expansions of Fourier transforms and discrete polyharmonic Green's functions, Pac. J. Math. 20 (1967) 85–98.
[14] P.G. Martinsson, G.J. Rodin, Asymptotic expansions of lattice Green's functions, Proc. R. Soc. Lond. Ser. A, Math. Phys. Sci. 458 (2002) 2609–2622.
[15] P.G. Martinsson, G.J. Rodin, Boundary algebraic equations for lattice problems, Proc. R. Soc. Lond. Ser. A, Math. Phys. Sci. (2009).
[16] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325–348.
[17] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numer. 6 (1997) 229–269.
[18] Z. Gimbutas, V. Rokhlin, A generalized fast multipole method for nonoscillatory kernels, SIAM J. Sci. Comput. 24 (2003) 796–817.
[19] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, J. Comput. Phys. 196 (2004) 591–626.
[20] P.G. Martinsson, V. Rokhlin, An accelerated kernel-independent fast multipole method in one dimension, SIAM J. Sci. Comput. 29 (2007) 1160–1178.
[21] W. Fong, E. Darve, The black-box fast multipole method, J. Comput. Phys. 228 (2009) 8712–8725.
[22] B. Zhang, J. Huang, N.P. Pitsianis, X. Sun, A Fourier-series-based kernel-independent fast multipole method, J. Comput. Phys. 230 (2011) 5807–5821.
[23] A. Dutt, M. Gu, V. Rokhlin, Fast algorithms for polynomial interpolation, integration, and differentiation, SIAM J. Numer. Anal. 33 (1996) 1689–1711.
[24] C. Berman, Grid-multipole calculations, SIAM J. Sci. Comput. 16 (1995) 1082–1091.
[25] J.R. Phillips, J.K. White, A precorrected-FFT method for electrostatic analysis of complicated 3-d structures, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 16 (1997) 1059–1072.
[26] P. Chatelain, P. Koumoutsakos, A Fourier-based elliptic solver for vortical flows with periodic and unbounded directions, J. Comput. Phys. 229 (2010) 2425–2431.
[27] L.N. Trefethen, Spectral Methods in MATLAB, vol. 10, SIAM, Philadelphia, PA, 2000.
[28] A. Quarteroni, F. Saleri, P. Gervasio, Scientific Computing with MATLAB and Octave, Springer, Berlin, 2010.

[29] S.G. Johnson, M. Frigo, A modified split-radix FFT with fewer arithmetic operations, IEEE Trans. Signal Process. 55 (2007) 111–119.
[30] G.M. Morton, A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing, IBM, 1966.
[31] H. Langston, L. Greengard, D. Zorin, A free-space adaptive fmm-based pde solver in three dimensions, Commun. Appl. Math. Comput. Sci. 6 (2011) 79–122.
[32] L. Ying, G. Biros, D. Zorin, H. Langston, A new parallel kernel-independent fast multipole method, in: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003, p. 14.
[33] A. Gholaminejad, D. Malhotra, H. Sundar, G. Biros, Fft, fmm, or multigrid? A comparative study of state-of-the-art Poisson solvers, in: Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium, 2014.
[34] J. Cserti, Application of the lattice Green's function for calculating the resistance of an infinite network of resistors, Am. J. Phys. 68 (2000) 896.
[35] M. Abramowitz, I.A. Stegun, Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables, Dover, New York, 1972.
[36] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, vol. 1, MIT Press, 1999.