A version of the fast multipole method (FMM) is described for charge distributions on the line. Previously published schemes of this type relied either on analytical representations of the potentials to be evaluated (multipoles, Legendre expansions, Taylor series, etc.), or on the Singular Value Decomposition (SVD); in contrast, the algorithm of this paper utilizes the matrix compression scheme described in *H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin. "On the compression of low rank matrices" SIAM J. Sci. Comput. 26(4):1389–1404, 2005.*, resulting in substantial improvements in the CPU time requirements. Furthermore, the scheme of this paper is applicable to a wide variety of potentials; in this respect, it is similar to the SVD-based FMMs. The performance of the scheme is illustrated with several numerical examples.

An accelerated kernel-independent fast multipole method in one dimension

P.G. Martinsson[†], V. Rokhlin[‡]

Technical Report YALEU/DCS/TR-1353
May 10, 2006

This research was supported in part by AFOSR, under contract #FA950-C-05-0064, and in part by DARPA, under contract #HR0011-05-1-0002.

[†] Dept. of Applied Mathematics, Univ. of Colorado at Boulder, Boulder, CO 80309-0526
[‡] Dept. of Mathematics, Yale University, New Haven CT 06511

Keywords: *Fast summation, fast multipole method, skeletonization.*

An accelerated kernel-independent fast multipole method in one dimension

P.G. Martinsson and V. Rokhlin

1. Introduction

We consider the problem of rapidly evaluating the sum

$$u_m = \sum_{n=1}^{N} K(x_m, x_n)\, q_n, \qquad \text{for } m = 1, \ldots, N, \tag{1.1}$$

given a set of points $(x_n)_{n=1}^N$ in $\mathbb{R}$, a kernel K that is smooth away from the diagonal, and a set of real or complex numbers $(q_n)_{n=1}^N$. We refer to the numbers q_n as "sources", the numbers u_n as "potentials", and the points x_n as "source locations".

The sum (1.1) can be straight-forwardly evaluated using $O(N^2)$ floating point operations. In many applications, this cost is prohibitively large, and a number of methods that reduce the cost to $O(N)$ or $O(N \log^\kappa N)$ have been developed, [1, 8]. In this paper, we describe an $O(N)$ method that is a development of the fast multipole method [8, 9]. The changes introduced enable the application of the method to a large class of kernels, and makes the method significantly faster than existing methods for the case where the sum (1.1) has to be evaluated several times for a fixed set of locations $\{x_n\}_{n=1}^N$ (see Section 6).

There are two principal differences between the method presented here and the original fast multipole method of [8]. The first concerns the way sources and potentials are represented. While the fast multipole method relies on an analytic properties of the kernel, the method presented here relies on a pre-computation step that adaptively constructs representations that are optimized for the given kernel, and the given source locations. The second difference concerns the treatment of the interactions between adjacent sub-intervals on the finest level of sub-division. While the original fast multipole method evaluates such interactions directly, the representations used by the method of this paper enables the compression of such interactions.

The method presented here is similar to the methods of [7, 14, 23] in that they all rely on adaptively computed representations, and thus work for a wide range of kernels. While the methods of [7, 23] rely on the singular value decomposition to compress the kernel, and [14] relies on the QR-decomposition, this paper is based on a technique described in [4, 15, 16] that we refer to as "skeletonization". Similar techniques were previously used in [17, 18, 22].

This paper is structured as follows: Section 2 lists some results from numerical linear algebra that will be of use. Section 3 describes the adaptive technique for the representation of sources and potentials that the current method relies on. Section 4 describes a fast summation scheme based on

the same data structures as the original fast multipole method, but using the representation described in Section 3. Section 5 describes an additional acceleration of the method based on compressing interactions between *all* boxes, including adjacent ones. Section 6 reports the results of several numerical experiments, and Section 7 summarizes the results.

Remark 1. This paper deals with the one-dimensional case. Extensions of the method to higher dimensions are under investigation and will be reported at a later date.

2. Preliminaries

The fast summation technique described in this paper achieves acceleration by approximating off-diagonal blocks of the matrix $[K(x_m, x_n)]_{m,n=1}^{N}$ in (1.1) by low-rank matrices. The particular matrix factorization we use to represent the low-rank matrix was described in [12] and [4]. The following lemma summarizes the facts needed for our purposes.

Lemma 1. *Let A be an $M \times N$ matrix of rank k with columns $C_1, \ldots, C_N$ and rows $R_1, \ldots, R_M$, so that*

$$A = [C_1 \cdots C_N] = \begin{bmatrix} R_1 \\ \vdots \\ R_M \end{bmatrix}.$$

Then

$$A = A_{\text{col}} \circ \text{proj}, \tag{2.1}$$

where proj *is a $k \times N$ matrix that contains the $k \times k$ identity as a submatrix, and where A_{col} is an $M \times k$ matrix consisting of k columns of A,*

$$A_{\text{col}} = [C_{n_1}, \ldots, C_{n_k}].$$

Furthermore,

$$A = \text{eval} \circ A_{\text{row}}, \tag{2.2}$$

where eval *is an $M \times k$ matrix that contains the $k \times k$ identity as a submatrix, and where A_{row} is a $k \times N$ matrix consisting of k rows of A,*

$$A_{\text{row}} = \begin{bmatrix} R_{m_1} \\ \vdots \\ R_{m_k} \end{bmatrix}.$$

Moreover, no elements of eval *or* proj *have magnitude larger than* 1.

Remark 2. Since the matrix proj contains a $k \times k$ identity matrix it can be applied to a vector using $k \times (N-k)$ multiplications and additions. Similarly, the matrix eval can be applied to a vector using $k \times (M-k)$ multiplications and additions.

Remark 3. As a direct consequence of Lemma 1, the condition number of proj is bounded by $\sqrt{1+k(N-k)}$ and the condition number of eval is bounded by $\sqrt{1+k(M-k)}$.

Remark 4. For simplicity, Lemma 1 is stated only for the case where the matrix A has exact rank k. Similar factorizations can be constructed for matrices of approximate rank k, see [4].

Remark 5. Methods for computing the factorizations in Lemma 1 are described in [4, 12]. The computational cost is typically $O(MNk)$ but can in rare cases be slightly higher.

3. Representation of functions via tabulation

3.1. **Outline.** At the core of the original fast multipole method is a technique for compactly representing sources and potentials. A source distribution inside a box is represented by a multipole expansion about the center of the box. From this expansion, the potential caused by the source distribution at distant target points can be evaluated. For a given accuracy, only a small number of terms in the expansion are needed, regardless of how many sources made up the original distribution. In the same way that source distributions are efficiently represented via multipole expansions, potentials are represented by giving the expansion coefficients in an expansion in harmonic polynomials.

In this section, we describe an alternative technique for representing sources and potentials. In order to describe the technique, we consider a simple model problem: Assume that we are given N source locations $(y_n)_{n=1}^N$ in a set b, M target locations $(x_m)_{m=1}^M$ in a set a, an interaction kernel $K(x,y)$, and that for a given vector of sources $q^b = (q_n)_{n=1}^N$, we wish to determine the vector of potentials $u^a = (u_m)_{m=1}^M$ given by

$$u^a = \text{direct}(a,b)\, q^b, \tag{3.1}$$

where direct(a,b) is the $M \times N$ matrix with entries $K(x_m, y_n)$. We demonstrate that if the matrix direct(a,b) has rank k (to within some precision ε), then it is possible to choose a subset of k source locations $(y_{n_j})_{j=1}^k$ with the property that the potential u^a can be replicated at all source points by placing some "proxy" sources on the points y_{n_j}. These proxy sources form the representation of the original source distribution. Similarly, it is possible to choose a subset of k target locations $(x_{m_i})_{i=1}^k$ with the property that if the potential is known at these k points, then it can be interpolated to all the remaining points. The potentials $(u_{m_i})_{i=1}^k$ form the representation for the potential u^a.

The representation technique described in this section has two principal advantages over techniques based on analytic methods, such as multipole expansions: (1) It is cheaper to construct the proxy sources that represent a source distribution than it is to compute the corresponding multipole expansion (or any other similar representation). (2) The operator that maps

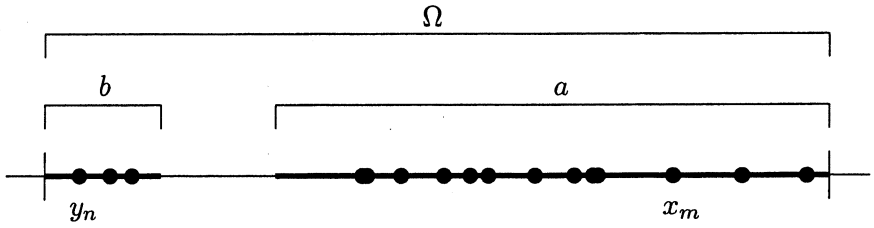

FIGURE 1. The computational domain described in Section 3.2.

a representation for a source distribution to a representation for a potential (sometimes called a "translation operator") is a submatrix of the original matrix of interaction direct(a, b). As a result, this operator need never be separately constructed or stored.

3.2. **Notation.** We assume that we are given a computational domain Ω, containing a number of source locations. For a given subset $b \subset \Omega$, we let $(y_n)_{n=1}^N$ denote the locations inside b, and we let $q^b = (q_n)_{n=1}^N$ denote a set of sources located at the points $(y_n)_{n=1}^N$. We let a denote the set of all points x that are *well-separated* from b, meaning that $\text{dist}(x, b) \geq \text{diam}(b)$, where $\text{diam}(b) = \sup_{y,y' \in b} |y - y'|$. We let $(x_m)_{m=1}^M$ denote the locations inside a, and let $u^a = (u_m)_{m=1}^M$ denote the potential on a induced by the charges q^b (so that u^a and q^b satisfy (3.1)). See Figure 1.

3.3. **Construction of the representation.** We first consider the task of evaluating a potential in a caused by a set of charges in b. To this end, we form the $M \times N$ matrix $A = \text{direct}(a, b)$ with entries $K(x_m, y_n)$, we determine its ε-rank k, and form a $k \times N$ matrix $\text{proj}(b)$, and an index vector $(n_j)_{j=1}^k$ such that, *cf.* (2.1),

$$A = A_{\text{col}} \circ \text{proj}(b) + O(\varepsilon),$$

where A_{col} is the $M \times k$ matrix whose j'th column is the n_j'th column of A. Then given any charge distribution q^b on b, we form the vector

$$\psi^b = \text{proj}(b)\, q^b \in \mathbb{C}^k. \tag{3.2}$$

The vector ψ^b has the property that the potential in a caused by the charge distribution q^b can to within precision ε be reconstructed from ψ^b, since

$$u^a = A\, q^b = \big(A_{\text{col}} \circ \text{proj}(b) + O(\varepsilon)\big)\, q^b = A_{\text{col}}\, \psi^b + O(\varepsilon). \tag{3.3}$$

We say that ψ^b is the *outgoing representation* of q^b, and that the points $(y_{n_j})_{j=1}^k$ form the *outgoing skeleton* of b.

We next consider the task of evaluating a potential in b caused by a charge distribution in a. To this end, we form the $N \times M$ matrix $B = \text{direct}(b, a)$

with entries $K(y_n, x_m)$, we determine its ε-rank k, and form an $N \times k$ matrix $\text{eval}(b)$, and an index vector $(n_i)_{i=1}^k$ such that

$$B = \text{eval}(b) \circ B_{\text{row}} + O(\varepsilon),$$

where B_{row} is the $k \times M$ matrix whose i'th row is the n_i'th row of B. Then given any distribution q^a of charges on a, we form the vector

$$\phi^b = B_{\text{row}}\, q^a \in \mathbb{C}^k. \tag{3.4}$$

The vector ϕ^b has the property that the potential on b can to within precision ε be reconstructed from ϕ^b only, since

$$u^b = B\, q^a = \big(\text{eval}(b) \circ B_{\text{row}} + O(\varepsilon)\big)\, q^q = \text{eval}(b)\, \phi^b + O(\varepsilon). \tag{3.5}$$

We say that ϕ^b is the *incoming representation* of u^b, and that the points $(y_{n_i})_{i=1}^k$ form the *incoming skeleton* of b.

Remark 6. The numbers in the vectors ψ^b and φ^b admit simple heuristic interpretations: Writing out equation (3.3) componentwise, we find that

$$u_m^a = \sum_{j=1}^{k} K(x_m, y_{n_j})\, \psi_j^b + O(\varepsilon), \qquad m = 1, \ldots, M. \tag{3.6}$$

In other words, the numbers ψ_j^b can be interpreted as charges that replicate the potential u^a when placed at the skeleton points y_{n_j}. Analogously, writing out equation (3.4) componentwise, we find that

$$\phi_i^b = \sum_{m=1}^{M} K(y_{n_i}, x_m)\, q_m^a + O(\varepsilon) = u_{n_i}^b + O(\varepsilon), \qquad i = 1, \ldots, k. \tag{3.7}$$

In other words, the number ϕ_i^b is simply the potential at the point y_{n_i}.

Remark 7. For many kernels it is possible to prove that when two sets a and b are separated by some finite distance, there exist basis functions $\{f_j\}_{j=1}^p$ and $\{g_j\}_{j=1}^p$ such that

$$\sup_{(x,y) \in a \times b} |K(x,y) - \sum_{j=1}^{p} f_j(x) g_j(y)| \leq \varepsilon, \tag{3.8}$$

where p tends to scale as a small power of $|\log \varepsilon|$ as $\varepsilon \to 0$. When (3.8) holds, the number p provides an upper bound on the numerical rank of the matrix of interaction $\text{direct}(a, b)$, regardless of the number of target and source points and their locations. Moreover, when a formula like (3.8) can be constructed using analytical properties of the kernel, it can be used to accelerate the computation of proj and eval, *cf.* Section 3.6.

Remark 8. In most environments, it is possible to construct the representations (3.3) and (3.5) in such a manner that the outgoing and the incoming skeletons are identical, see [4]. This causes only a minor increase in the number p.

3.4. **Converting outgoing to incoming representations.** In this section, we construct a matrix that converts the outgoing representation for a set of charges in one box, to an incoming representation for the potential induced by this set of charges in a different box.

Given two well-separated boxes a and b in Ω, suppose that we are given the outgoing representation $\psi^b = (\psi_j)_{j=1}^{k_b}$ for a charge distribution q^b in b, and that we wish to determine the incoming representation ϕ^a for the potential u^a induced by q^b. We let $(x_m)_{m=1}^M$ and $(y_n)_{n=1}^N$ denote the locations in a and b, and we let $(x_{m_i})_{i=1}^{k_a}$ and $(y_{n_j})_{j=1}^{k_b}$ denote the corresponding skeletons. Then equation (3.7) implies that $\phi_i^a = u_{m_i}^a + O(\varepsilon)$, and equation (3.6) implies that $u_{m_i}^a = \sum_{j=1}^{k_b} K(x_{m_i}, y_{n_j})\, \psi_j^b + O(\varepsilon)$. In other words, letting $\mathrm{oi}(a,b)$ denote the $k_a \times k_b$ matrix with entries $K(x_{m_i}, y_{n_j})$, we find that

$$\phi^a = \mathrm{oi}(a,b)\, \psi^b + O(\varepsilon). \tag{3.9}$$

We refer to the matrix $\mathrm{oi}(a,b)$ as an *outgoing-to-incoming* translation operator. This operator is a submatrix of $\mathrm{direct}(a,b)$. One ramification of this fact is that $\mathrm{oi}(a,b)$ need not be explicitly constructed. Since the kernel $K(x,y)$ is known, $\mathrm{oi}(a,b)$ is uniquely defined by the incoming and outgoing skeletons of a and b.

Remark 9. The matrix $\mathrm{direct}(a,b)$ is related to the matrices $\mathrm{eval}(a)$, $\mathrm{oi}(a,b)$, and $\mathrm{proj}(b)$ via the relation

$$\mathrm{direct}(a,b) = \mathrm{eval}(a) \circ \mathrm{oi}(a,b) \circ \mathrm{proj}(b) + O(\varepsilon). \tag{3.10}$$

To prove (3.10), we first note that $\mathrm{direct}(a,b)$ is defined by the relation

$$u^a = \mathrm{direct}(a,b)\, q^b. \tag{3.11}$$

Moreover, (3.5), (3.9), and (3.2), imply that

$$u^a = \mathrm{eval}(a)\, \phi^a + O(\varepsilon), \tag{3.12}$$

$$\phi^a = \mathrm{oi}(a,b)\, \psi^b + O(\varepsilon), \tag{3.13}$$

$$\psi^b = \mathrm{proj}(b)\, q^b, \tag{3.14}$$

respectively. Since (3.11) must hold for every q^b, the equations (3.11), (3.12), (3.13), and (3.14) together imply (3.10). (We remark that equation (3.10) is analogous to equation (3.1) in [4].)

3.5. **Merging the representations of two boxes.** In this section we describe a procedure for the construction of the outgoing representation of a set b if the outgoing representations are known for two sets b_1 and b_2 such that $b = b_1 \cup b_2$.

We let a denote the set of points that are well-separated from b, and for $j = 1, 2$, we similarly let a_j denote the set of points that are well-separated from b_j. It follows that $a \subseteq a_j$. Typically, a is strictly smaller than a_j.

We let u^a denote the potential induced on the locations $(x_m)_{m=1}^M \subset a$ by two given charge distributions in b_1 and b_2 with outgoing representations $\psi^{(1)}$

and $\psi^{(2)}$. Furthermore, we let $y^{(1)}$ and $y^{(2)}$ be the corresponding outgoing skeletons. (We recall that $y^{(1)}$ and $y^{(2)}$ are subsets of the original sets of locations in b_1 and b_2.) Merging these two pairs of vectors, we form $\hat{\psi} = [\psi^{(1)}, \psi^{(2)}]$, and $\hat{y} = [y^{(1)}, y^{(2)}]$. Since a is wholly contained in $a_1 \cap a_2$, equation (3.6) implies that

$$u^a_m = \sum_{j=1}^{k_1+k_2} K(x_m, \hat{y}_j)\, \hat{\psi}_j + O(\varepsilon), \qquad \text{for } m = 1, \ldots, M, \tag{3.15}$$

where k_1 and k_2 are the lengths of $\psi^{(1)}$ and $\psi^{(2)}$. We rewrite equation (3.15) as a matrix-multiplication by introducing the $M \times (k_1 + k_2)$ matrix A with entries $A_{mj} = K(x_m, \hat{y}_j)$,

$$u^a = A\,\hat{\psi} + O(\varepsilon). \tag{3.16}$$

Equation (3.15) says that $\hat{\psi}$ is a valid outgoing representation for b with an associated outgoing skeleton $\hat{y}$. However, when a contains fewer target locations than $a_1 \cup a_2$ (which is typically the case), this representation is likely to be longer than necessary. To be precise, the length of an optimal representation equals the ε-rank of the matrix A in (3.16), which we denote by k. To obtain an optimal representation, we factor A as in (2.1),

$$A = \tilde{A}_{\text{col}} \circ Z + O(\varepsilon),$$

where Z is a $k \times (k_1 + k_2)$ matrix, and $\tilde{A}_{\text{col}}$ consists of k columns of A. We denote the indices of these by $(j_i)_{i=1}^k$. An optimal outgoing representation is then the vector $\psi^b = Z\hat{\psi}$; it is associated with the outgoing skeleton $(\hat{y}_{j_i})_{i=1}^k$. Letting $\text{oo}(b, b_1)$ denote the matrix formed by the first k_1 columns of Z, and letting the remaining columns form $\text{oo}(b, b_2)$, we find that

$$\psi^b = \text{oo}(b, b_1)\, \psi^{(1)} + \text{oo}(b, b_2)\, \psi^{(2)}.$$

Analogously, we construct matrices $\text{ii}(b_1, b)$ and $\text{ii}(b_2, b)$ that construct the incoming representations $\phi^{(1)}$ and $\phi^{(2)}$ for b_1 and b_2 from the incoming representation of b via the formulas

$$\phi^{(1)} = \text{ii}(b_1, b)\, \phi^b, \qquad \text{and} \qquad \phi^{(2)} = \text{ii}(b_2, b)\, \phi^b. \tag{3.17}$$

We refer to the matrix $\text{oo}(b, b_1)$ as an *outgoing-to-outgoing* translation operator, while the matrix $\text{ii}(b_1, b)$ is referred to as an *incoming-to-incoming* translation operator.

3.6. **Efficient construction of translation operators.** The techniques for constructing translation operators that were given in Sections 3.3, 3.4, and 3.5 could potentially be quite expensive. For instance, when computing the operator $\text{proj}(b)$ in Section 3.3, we considered the interaction between the set of locations in b, and the set of all locations in Ω that are well-separated from b. The second set is typically very large. However, in most instances of practical interest, the process can be accelerated by replacing this very large set, by a small set of pre-determined locations in a that act

as proxies for the actual charge locations. This acceleration technique works for kernels satisfying the following:

Assumption I: *Let b denote a subset of a computational domain Ω, and let a denote the set of points in Ω that are well-separated from b. We assume that for any positive number ε, there exist interpolation points $(z_i)_{i=1}^p \subset a$ and functions $(\varphi_i)_{i=1}^p$ such that*

$$\sup_{x\in a}\sup_{y\in b}|K(x,y)-\sum_{i=1}^{p}\varphi_i(x)K(z_i,y)|\leq\varepsilon. \tag{3.18}$$

The number of terms required, p, is assumed to satisfy

$$p\leq C|\log\varepsilon|,$$

as $\varepsilon\to 0$. For non-symmetric kernels, we also assume that there exist points $(w_i)_{i=1}^p \subset a$ and functions $(\psi_i)_{i=1}^p$ such hat

$$\sup_{x\in a}\sup_{y\in b}|K(y,x)-\sum_{i=1}^{p}K(y,w_i)\psi_i(x)|\leq\varepsilon. \tag{3.19}$$

Remark 10. It is shown in [16] that Assumption I holds for any kernel that is separable in the sense described in Remark 7.

Suppose now that the kernel K satisfies Assumption I. Then for for any given set $b \subset \Omega$, one can quite inexpensively construct an outgoing representation that is valid for evaluating potentials at the points z_i. Assumption I then assures us that this representation is also valid at any other point that is well-separated from b, since the potential there can be interpolated locally from the potential at the z_i's. Incoming representations can be constructed analogously.

Remark 11. In general, the cost of constructing the matrix $\mathrm{proj}(b)$ and determining the outgoing skeleton $(y_{n_j})_{j=1}^k$, is $O(kNM)$ where k is the ε-rank of interaction, N is the number of points in b, and M is the number of locations that are well-separated from b. When the kernel K satisfies Assumption I, this cost can be reduced to $O(kNp)$, with no dependence on M.

3.7. **Representations with extended domains of validity.** For a given box b, we have so far constructed outgoing and incoming representations that are valid with respect to locations in the computational domain that are separated from a box b by at least the diameter of b. This requirement of a "buffer" zone is a standard feature of fast summation techniques. However, the particular technique for representing functions described in Section 3.3 can also be used to compress the interaction between *adjacent* boxes. The resulting representations are very similar to the ones constructed for the interactions between separated boxes. However, the expansions are longer, and should not be used unless necessary. We call such unbuffered

representations "rich representations", and denote the outgoing and the incoming rich representations by Ψ^b, and Φ^b, respectively. The corresponding projection and evaluation operators are called Proj, and Eval (capitalized), respectively, so that for a given box b

$$\Psi^b = \mathrm{Proj}(b)\, q^b, \tag{3.20}$$

$$u^b = \mathrm{Eval}(b)\, \Phi^b. \tag{3.21}$$

The representation Ψ^b in (3.20) can be used to compute the potential induced by q^b at any location outside b, while the representation Φ^b may be used to reconstruct any potential u^b induced by a distribution of charges at any of the locations outside b.

Remark 12. In the original FMM, and its variants, the existence of a buffer between source and target boxes was necessary to assure that the classical representations (multipole series, etc.) are valid, and have controlled convergence rates. Here, we eliminate the need for the buffers by constructing the representations via numerical techniques (as opposed to using analytic properties of the kernel). Such representations are possible because they are not required to be valid everywhere in the boxes, but only at a finite number of source and target points.

Remark 13. If for a given box b, we wish to compute both its rich and its regular outgoing representations, the most efficient way for doing so is to first compute the rich representation Ψ^b, and then compute the regular one, ψ^b, from it.

In order to construct the matrix that maps Ψ^b to ψ^b, we first construct the matrix A with entries $A_{mj} = K(x_m, y_j)$, where $(y_j)_{j=1}^K$ are the points in the rich skeleton, and $(x_m)_{m=1}^M$ is a set of well-separated target points. Letting k denote the ε-rank of A, we factor A as in (2.1),

$$A = A_{\mathrm{col}} \circ \overline{\mathrm{proj}(b)} + O(\varepsilon),$$

where A_{col} is the $M \times k$ matrix consisting of the columns of A with indices $(j_i)_{i=1}^k$, and $\overline{\mathrm{proj}(b)}$ is a $k \times K$ matrix. If Ψ^b is an outgoing representation associated with the rich outgoing skeleton $(y_j)_{j=1}^K$, then

$$\psi^b = \overline{\mathrm{proj}(b)}\, \Psi^b$$

is a regular outgoing representation for b associated with the outgoing skeleton $(y_{j_i})_{i=1}^k$.

Analogously, we construct an operator $\overline{\mathrm{eval}(b)}$ that constructs a rich incoming representation from a regular one:

$$\Phi^b = \overline{\mathrm{eval}(b)}\, \phi^b. \tag{3.22}$$

Other types of translation operators involving rich representations can be constructed, but are not needed for the purposes of this paper.

4. A KERNEL-INDEPENDENT FAST SUMMATION TECHNIQUE

4.1. **Problem formulation.** In this section, we consider the problem of evaluating the sum

$$u_i = \sum_{j=1}^{N} K(x_i, x_j)\, q_j, \qquad i = 1, \ldots, N, \tag{4.1}$$

given a set of real numbers $(x_i)_{i=1}^N$, a set of real or complex numbers $(q_j)_{j=1}^N$, and a kernel $K(x, y)$. We call the numbers x_i "locations", the numbers q_i "charges", and the numbers u_i "potentials". The numbers x_i are all contained in an interval Ω called the "computational domain".

We focus on the situation where the potentials $(u_i)_{i=1}^N$ are to be evaluated for a sequence of charge distributions $(q_i)_{i=1}^N$ associated with a single set of locations $(x_i)_{i=1}^N$. In this environment, we spend a moderate amount of computational effort on optimizing the representations and the various translation operators used for the given set of locations. Once this has been done, each potential evaluation can be performed rapidly.

The summation technique presented in this section follows the same template as earlier versions of the fast multipole method, [8, 10, 7, 5]. The only difference between these methods, and the method presented in this section is that a different representation for potentials and sources is used. In Section 5, a further acceleration of the method is described. This acceleration is obtained by compressing the interactions between adjacent regions.

4.2. **Tree structure.** Given a computational domain $\Omega = [x_{\text{left}}, x_{\text{right}}) \subset \mathbb{R}$, and a set of locations $(x_n)_{n=1}^N \subset \Omega$, we construct an adaptive partitioning of the domain into subintervals in such a way that no interval contains more than N_0 locations, for some given (small) integer N_0. We do this via a hierarchical subdivision process in which any interval holding more than N_0 points is split into two halves, and then the process is continued with each half that in turn contains more than N_0 points. If $a = [x^a_{\text{left}}, x^a_{\text{right}})$ is an interval resulting from this process, then

$$\begin{aligned} x^a_{\text{left}} &= x_{\text{left}} + (j-1)\, 2^{-l} \left(x_{\text{right}} - x_{\text{left}}\right), \\ x^a_{\text{right}} &= x_{\text{left}} + j\, 2^{-l} \left(x_{\text{right}} - x_{\text{left}}\right), \end{aligned}$$

for some number $l = 0, 1, 2, \ldots$, and some number $j = 1, 2, \ldots, 2^l$. The number l is called the "level" of the box a and denotes how many times Ω has been cut in half to reach a.

For every box a, we construct the following lists of other boxes:

$\mathrm{L}_{\mathrm{children}}(a)$: For a non-leaf box a, this is a list of all boxes b that are contained in a, and that are separated from a by one level only. If $b \in \mathrm{L}_{\mathrm{children}}(a)$, we say that b is a "child" of a, and that a is a "parent" of b. For a leaf box, $\mathrm{L}_{\mathrm{children}}(a)$ is empty.

$\mathrm{L}_{\mathrm{close}}(a)$: For a leaf-box a, this is a list of all leaf boxes b that are not well-separated from a. For a non-leaf box, $\mathrm{L}_{\mathrm{close}}(a)$ is empty.

$\mathrm{L}_2(a)$: The list of all boxes b on the same level as a that are well-separated from a, but whose parents are not separated from the parent of a. (This list is known as the "interaction list" in the original FMM.)

4.3. **Outgoing and incoming representations.** For any box a, we let the vector of charges inside the box be denoted by q^a, and with $\mathrm{proj}(a)$ the matrix defined in Section 3.3, we let the vector

$$\psi^a = \mathrm{proj}(a)\, q^a, \tag{4.2}$$

denote the outgoing representation of a. Similarly, we let u^a_{far} denote the potential on a caused by all charges that are well-separated from a. An incoming representation for a is a vector ϕ^a such that

$$u^a_{\mathrm{far}} = \mathrm{eval}(a)\, \phi^a + O(\varepsilon), \tag{4.3}$$

where the operator $\mathrm{eval}(a)$ is defined in Section 3.3.

Remark 14. In the original fast multipole method, the function of the vector ψ^a was performed by a vector of multipole coefficients representing q^a, and the function of ϕ^a was performed by a vector of expansion coefficients for u^a_{far} in a basis of harmonic polynomials.

4.4. **Hierarchical construction of representations.** Both the incoming and the outgoing representations can be computed recursively. Specifically, if a is any non-leaf box a, its outgoing representation can be computed from the outgoing representations of its children via the formula

$$\psi^a = \sum_{b \in \mathrm{L}_{\mathrm{children}}(a)} \mathrm{oo}(a,b)\, \psi^b, \tag{4.4}$$

where the matrices $\mathrm{oo}(a,b)$ are defined in Section 3.5. Similarly, if a is any box other than the root box, the incoming representation ϕ^a (representing the potential u^a_{far} caused by charges in all boxes that are well-separated from a) can be constructed by combining the incoming potential of its parent b with the outgoing potentials of all boxes in the interaction list of a via the formula

$$\phi^a = \mathrm{ii}(a,b)\, \phi^b + \sum_{c \in \mathrm{L}_2(a)} \mathrm{oi}(a,c)\, \psi^c, \tag{4.5}$$

where the list $L_2(a)$ is defined in Section 4.2, the matrix $\mathrm{ii}(a,b)$ is defined in Section 3.5, and the matrices $\mathrm{oi}(a,c)$ are defined in Section 3.4.

4.5. **Pre-computation.** Given a set of locations $(x_n)_{n=1}^N$ and a kernel function $K(x, y)$, the pre-computation stage consists of the following steps:

(1) Divide the computational domain into a tree structure, as described in Section 4.2.
(2) Construct the lists described in Section 4.2.
(3) Loop over all leaf boxes. For each box a, construct the regular outgoing and incoming skeletons. Simultaneously, determine the matrices $\text{proj}(a)$ and $\text{eval}(a)$, described in Section 3.3.
(4) Loop over all non-leaf boxes, going from finer to coarser levels. For each box a, construct the regular outgoing and incoming skeletons by merging the skeletons of its children b_i. Simultaneously, determine the matrices $\text{oo}(a, b_i)$, and $\text{ii}(b_i, a)$, as described in Section 3.5.
(5) Loop over all boxes a, and then over all elements b in the list $\text{L}_2(a)$. For each such pair (a, b), construct the translation operator $\text{oi}(a, b)$, as described in Section 5.3.

The total cost of the steps described above depends on the kernel and on the charge distribution. If the kernel satisfies Assumption I in Section 3.6, then the computational cost is typically either $O(N)$, or $O(N \log N)$, and the amount of storage required is typically $O(N)$, *cf.* Remark 15.

4.6. **A general fast multipole method.** We have now assembled the tools for computing the sum (4.1) through two passes through the hierarchical tree; one upwards, and one downwards.

(1) Sweep over all leaf boxes a. For each box, construct its outgoing representation from the values of the charges inside it, *cf.* (4.2):

$$\psi^a = \text{proj}(a)\, q^a.$$

(2) Sweep over all non-leaf boxes a, going from finer to coarser levels. For each box a, construct its outgoing representation by merging the outgoing representations of its children, *cf.* (4.4):

$$\psi^a = \sum_{b \in \text{L}_{\text{children}}(a)} \text{oo}(a, b)\, \psi^b.$$

(3) Set $\phi^r = 0$ for the root box r. Then loop over all boxes (including the root box), going from coarser to finer levels. For each box a, form its incoming representation by combining the incoming representation of its parent, box b, with the contributions from the boxes in $\text{L}_2(a)$ (its "interaction list"), *cf.* (4.5):

$$\phi^a = \text{ii}(a, b)\, \phi^b + \sum_{c \in \text{L}_2(a)} \text{oi}(a, c)\, \psi^c.$$

(4) Sweep over all leaf nodes a. For each node, form the potential u^a by evaluating the incoming representation and directly adding the

contributions from the charges inside a and in all boxes that are not well-separated from a, *cf.* (4.3):

$$u^a = \text{eval}(a)\,\phi^a + \text{direct}(a,a)\,q^a + \sum_{c \in \text{L}_{\text{close}}(a)} \text{direct}(a,c)\,q^c.$$

For kernels satisfying Assumption I, the computational cost of the steps described in this subsection is typically $O(N)$.

Remark 15. It is possible to construct (highly non-uniform) charge distributions for which the computational cost of the steps described in Sections 4.5 and 4.6 exceed $O(N)$. A discussion of this phenomenon, and a modification to the scheme that makes it always retain $O(N)$ complexity, are described in [19].

5. An accelerated kernel independent fast multipole method

5.1. **Outline.** In this section, we again consider the problem of rapidly evaluating the sum (4.1). We describe a technique for doing so that is similar to the technique described in Section 4, but with the difference that even interactions between adjacent boxes are compressed. To enable this additional compression, we keep track of four representations for each leaf box; the regular outgoing and incoming ones described in Section 3.3, and also the rich outgoing and incoming representations described in Section 3.7. The rich representations are used exclusively for the purpose of evaluating interactions involving at least one leaf box.

The asymptotic cost for the algorithm described in this section scales in the same way with N as the cost for the algorithm described in Section 4.6 (typically, $O(N \log N)$ for pre-computation, and $O(N)$ for evaluation). However, the constants involved are smaller.

5.2. **Tree structure.** The accelerated algorithm uses the same tree structure that was described in Section 4.2. In addition to the lists described in that section, the accelerated algorithm also uses the following three lists:

- $\text{L}_1(a)$: For a leaf box a, this is a list of the leaf boxes that directly border a. For a non-leaf box, $\text{L}_1(a)$ is empty.
- $\text{L}_3(a)$: For a leaf box a, this is a list of all boxes on finer levels than a that are separated from a but whose parents are not separated from the parent of a. For a non-leaf box a, $\text{L}_3(a)$ is empty.
- $\text{L}_4(a)$: The dual of L_3. In other words, $b \in \text{L}_4(a)$ if and only if $a \in \text{L}_3(b)$.

5.3. **Translation operators.** The algorithm described in Section 4.6 utilizes a single type of outgoing-to-incoming translation operator. This operator maps a regular outgoing representation ψ^a to a regular incoming representation ϕ^b for all pairs (a,b) such that $a \in \text{L}_2(b)$. This translation operator is also used in the accelerated algorithm described in this section, we label it $\text{oi}_2(a,b)$. The accelerated algorithm requires three additional outgoing-to-incoming translation operators: For each pair (a,b) such that

$b \in \mathrm{L}_1(a)$, the operator $\mathrm{oi}_1(a,b)$ maps a rich representation to a rich representation. For each pair (a,b) such that $b \in \mathrm{L}_3(a)$, the operator $\mathrm{oi}_3(a,b)$ maps a regular representation to a rich representation. For each pair (a,b) such that $b \in \mathrm{L}_4(a)$, the operator $\mathrm{oi}_4(a,b)$ maps a rich representation to a regular representation. Each matrix $\mathrm{oi}_j(a,b)$ is the restriction of the original matrix of interaction $\mathrm{direct}(a,b)$ to the relevant outgoing and incoming skeletons for a and b, respectively.

5.4. **Pre-computation.** The pre-computation for the accelerated algorithm is very similar to the pre-computation described in Section 4.5. The difference is that for all leaf nodes, we compute both the regular and the rich skeletons (and the corresponding operators $\overline{\mathrm{eval}}$ and $\overline{\mathrm{proj}}$). We also determine the additional lists L_1, L_3, and L_4, as well as the corresponding translation operators oi_1, oi_3, and oi_4.

5.5. **Potential evaluations.** After the particle locations $(x_n)_{n=1}^N$, and the kernel $K(x,y)$ have been fixed, and the pre-computation described in Section 5.4 has been completed, the following steps will compute the vector of potentials $(u_n)_{n=1}^N$ from a vector of charges $(q_n)_{n=1}^N$, (*cf.* (4.1)), to within a precision of computations ε:

(1) Loop over all leaf boxes. For each box a, compute Ψ^a and then ψ^a:

$$\Psi^a = \mathrm{Proj}(a)\, q^a, \quad \text{and then} \quad \psi^a = \overline{\mathrm{proj}(a)}\, \Psi^a.$$

(2) Loop over all non-leaf boxes, going from finer levels to coarser. For each box a, compute ψ^a by combining the outgoing representations of its children,

$$\psi^a = \sum_{b \in \mathrm{L}_{\mathrm{children}}(a)} \mathrm{oo}(a,b)\, \psi^b.$$

(3) Loop over all leaf boxes. For each box a, add up the contributions from the boxes in $\mathrm{L}_1(a)$,

$$\Phi_1^a = \sum_{b \in \mathrm{L}_1(a)} \mathrm{oi}_1(a,b) \Psi^b.$$

(4) Loop over all boxes. For each box a, add up the contributions from the boxes in $\mathrm{L}_2(a)$,

$$\phi_2^a = \sum_{b \in \mathrm{L}_2(a)} \mathrm{oi}_2(a,b) \psi^b.$$

(5) Loop over all leaf boxes. For each box a, add up the contributions from the boxes in $\mathrm{L}_3(a)$,

$$\Phi_3^a = \sum_{b \in \mathrm{L}_3(a)} \mathrm{oi}_3(a,b) \psi^b.$$

(6) Loop over all boxes. For each box a, add up the contributions from the boxes in $\mathrm{L}_4(a)$,

$$\phi_4^a = \sum_{b \in \mathrm{L}_4(a)} \mathrm{oi}_4(a,b) \Psi^b.$$

(7) Set $\phi^r = 0$ for the root box r. Then loop over all non-leaf boxes (including the root box), from coarser levels to finer. For each box a, and for each child b of a, construct the incoming regular representations for b:

$$\phi^b = \phi_2^b + \phi_4^b + \mathrm{ii}(b,a)\, \phi^a.$$

(8) Loop over all leaf boxes. For each box a, construct its rich incoming representation by adding the various contributions:

$$\Phi^a = \Phi_1^a + \Phi_3^a + \overline{\mathrm{eval}(a)}\, \phi^a.$$

(9) Loop over all leaf boxes. For each box a, construct u^a by interpolating Φ^a and adding the contributions from q^a:

$$u^a = \mathrm{Eval}(a)\, \Phi^a + \mathrm{direct}(a,a) q^a.$$

Remark 16. The vectors ϕ_2^a, ϕ_4^a, Φ_1^a, and Φ_3^a are never stored; they are added directly to either ϕ^a or Φ^a as they are computed.

6. Numerical examples

The numerical algorithm described in Section 5 has been implement in FORTRAN 77 and tested on several model problems. In this section, we summarize the results from tests involving three different kernels:

Example 1: This example involves the evaluation of the sum

$$u_m = \sum_{\substack{n=1 \\ n \neq m}}^{N} \left(\log |x_m - x_n|\right) q_n, \qquad \text{for } m = 1, \ldots, N, \tag{6.1}$$

where the points $(x_n)_{n=1}^N$ were drawn from a uniform random distribution on the interval [0, 1].

Example 2: This example involves the evaluation of the sum

$$u_m = \sum_{n=1}^{N} \frac{p_{k+1}(x_m)p_k(x_n) - p_k(x_m)p_{k+1}(x_n)}{x_m - x_n} q_n, \quad \text{for } m = 1, \ldots, N, \tag{6.2}$$

where p_k is the k'th Legendre polynomial, and the points $(x_n)_{n=1}^N$ are the Gaussian nodes on the interval $[-1, 1]$. The sum (6.2) arises in evaluating orthogonal projections onto the space of order k polynomials in $L^2([-1,1])$, see [13]. Similar sums are encountered in the construction of fast algorithms for the harmonic expansions on the sphere, in the FMM for the Helmholtz and Maxwell equations, etc., see [3, 20]. In the numerical examples reported, k was one third of N (rounded to the nearest integer).

	Matrix elements computed on the fly	Matrix elements pre-computed
Example 1	11	52
Example 2	20	60
Example 3	14	94

TABLE 1. Breakeven points extrapolated from Figure 2.

Example 3: This example involves the evaluation of the sum

$$u_m = \sum_{n=1}^{N} \frac{\sin(a(x_m - x_n))}{x_m - x_n} q_n, \qquad \text{for } m = 1, \ldots, N, \tag{6.3}$$

where $(x_n)_{n=1}^N$ are equispaced nodes in the interval $[-1, 1]$. Such sums occur frequently in signal processing and many other areas, see [2, 6, 21]. The parameter a was chosen so that there were 5 nodes per wavelength, *i.e.* $a = \pi N/5$.

The CPU times required for the accelerated matrix-vector multiplication in the three examples are given in Figure 2. The experiments were carried out on a 3.2GHz Pentium IV desktop with 2Gb of RAM. All calculations shown were carried out with a requested accuracy of $\varepsilon = 10^{-10}$. Detailed CPU time requirements are given in Tables 2, 3, and 4 in Appendix A. These tables also report the memory requirements of the algorithm, as well as the time required for the pre-computing step.

For comparison, Figure 2 also reports the CPU times required for uncompressed matrix-vector multiplies, both for the case where the matrix elements are pre-computed and stored, and the case where the matrix elements are computed on the fly. The break-even points obtained by extrapolating the lines in Figure 2 are given in Table 1. Finally, Figure 2 reports the CPU time requirement for an FFT of length N (with equispaced nodes). We note that for large problems, the matrix-vector multiply reported here is about 5 – 10 times slower than an FFT.

In order to investigate the dependence of the CPU time requirement on the requested accuracy, the matrix-vector multiplication in Example 1 was carried out for $\varepsilon = 10^{-3.5}$, 10^{-7}, and 10^{-14}. The resulting CPU times are reported in Figure 3. This table also specifies the CPU times required for the pre-computational stage. We remark that no attempt whatsoever was made to optimize this part of the code, and its CPU time requirements can be reduced dramatically.

7. CONCLUSIONS

This paper describes a fast multipole method for the rapid evaluation of sums of the form (1.1). In one and two dimensions, the computational complexity of this method is $O(N)$.

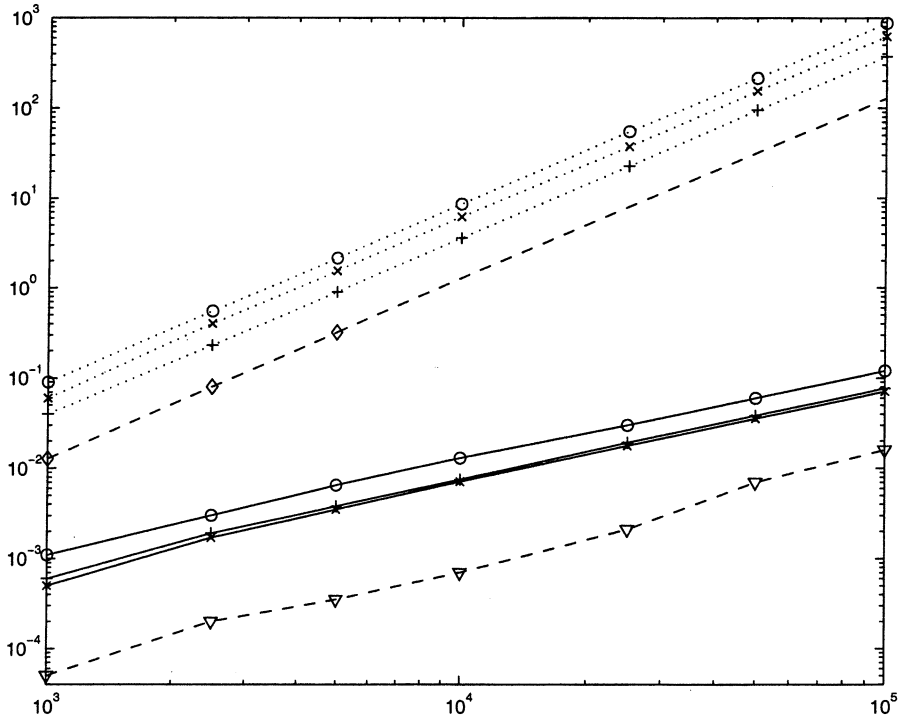

FIGURE 2. The CPU times (in seconds) required for evaluating the sums (6.1), (6.2), and (6.3) versus problem size, N. The markers '$\times$', '$+$', and '$\circ$' label the three cases. The dotted lines mark the times required for uncompressed matrix-vector multiplies, the solid lines mark the times required for the algorithm of Section 5. The times for a matrix-vector multiply with a stored matrix are marked with '$\diamond$' and the times for an FFT are marked '$\triangledown$'.

The method does not use any analytic expansions of the kernel; instead, it numerically compresses the kernel in a pre-processing stage whose computational cost is $O(N \log N)$. The combined cost of the pre-computation, and a single potential evaluation using the present scheme is larger than the cost of a single evaluation using some existing pre-computation free fast summation techniques. However, if the evaluation is to be performed for a sequence of different charge distributions (on a fixed set of charge locations), then the current method outperforms most existing methods, with the important exception of convolutional sums that can be evaluated using the FFT. Moreover, since the current scheme does not explicitly rely on analytical properties of the kernel, it is applicable in many environments in which pre-computation free methods are not available.

One application that seems particularly well suited for the fast summation technique of this paper concerns the rapid computation of the singular value

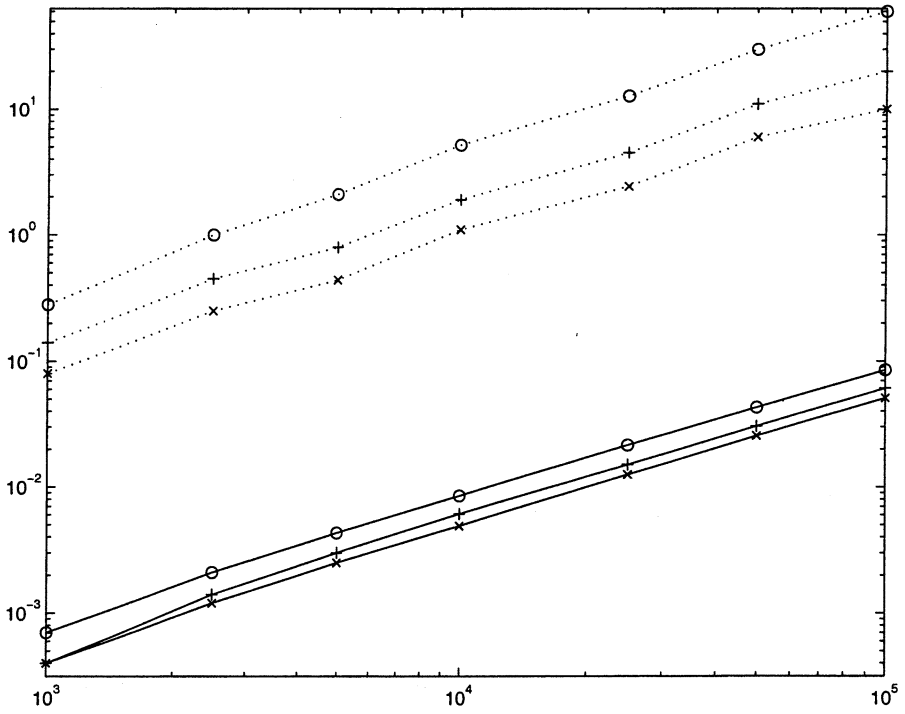

FIGURE 3. The CPU time required for evaluating the sum in (6.1) with $\varepsilon = 10^{-3.5}$, 10^{-7}, and 10^{-14}, using the algorithm of Section 5, plotted against problem size, N. The solid lines give the time for a matrix-vector multiply, while the dotted lines give the time required for pre-computation.

decomposition of a matrix. A very fast algorithm for this task based on a divide-and-conquer technique is described in [11]. A core observation of [11] is that the seemingly expensive task of updating a unitary matrix can – surprisingly – be accelerated using the fast multipole method. However, the break-even point of previous versions of the fast multipole method made the algorithm of [11] competitive only for very large matrices. Research into combining the fast summation technique of this paper with the algorithm of [11] is currently under way.

Acknowledgments: The authors wish to thank M. Tygert for constructive discussions.

APPENDIX A. COMPUTATIONAL RESULTS

This appendix contains detailed statistics for the computational experiments summarized in Section 6. The numbers upon which Figure 2 is based can be found in Tables 2, 3, and 4. The numbers upon which Figure 3 is based can be found in Table 5. Table 6 provides the CPU times required

for an uncompressed matrix-vector multiply, and a length-N FFT. The implementation of the FFT is the one found in FFTPACK.

The following numbers are given for each experiment:

N	Problem size.
ε	Requested accuracy.
E_{max}	Maximum error (see Remark 17).
E_{rms}	Root mean square error (see Remark 17).
t_{pre}	CPU time required for pre-computation.
t_{eval}	CPU time required for an accelerated matrix-vector multiply.
M_{keep}	Amount of memory required to store the compressed operator.
M_{pre}	Amount of memory required for pre-computation.
t_{uncomp}	CPU time required for an uncompressed matrix-vector multiply.
t_{matvec}	CPU time required for multiplying a stored matrix by a vector.
t_{fft}	CPU time required for a length-N (equispaced) FFT.

All CPU times are given in seconds, and all memory requirements are given in terms of storage for double precision reals. The memory required for applying the operator to a vector is not reported since it is far smaller than M_{keep} (asymptotically, it is a couple of reals per node).

Remark 17. We report both the maximum error E_{max} and the root mean square error E_{rms}. Letting $(u_n)_{n=1}^N$ denote the result of an uncompressed matrix-vector multiply, and letting $(u'_n)_{n=1}^N$ denote the result of an accelerated matrix-vector multiply, we have

$$\text{(A.1)} \quad E_{\text{max}} = \frac{\max_{1\leq n\leq N} |u_n - u'_n|}{(1/N)\sum_{n=1}^N |u_n|}, \quad \text{and} \quad E_{\text{rms}} = \sqrt{\frac{\sum_{n=1}^N (u_n - u'_n)^2}{\sum_{n=1}^N |u_n|^2}}.$$

In each of the experiments reported, the charges q_n were drawn from a uniform random distribution on $[-1, 1]$.

N	t_{pre}/N	$E_{\max}$	E_{rms}	t_{eval}/N	M_{keep}/N	M_{pre}/N	t_{uncomp}/N
1000	1.8E-04	2.3E-10	3.0E-11	5.0E-07	9.9E+01	1.4E+03	7.0E-05
2500	2.6E-04	2.5E-10	2.6E-11	6.8E-07	1.0E+02	7.5E+02	1.6E-04
5000	2.5E-04	3.9E-10	3.1E-11	7.0E-07	1.0E+02	4.9E+02	3.1E-04
10000	3.2E-04	2.4E-10	2.9E-11	7.1E-07	1.0E+02	3.3E+02	6.2E-04
25000	3.0E-04	2.4E-10	2.7E-11	7.1E-07	1.1E+02	2.1E+02	1.5E-03
50000	3.6E-04	3.0E-10	2.5E-11	7.1E-07	1.1E+02	1.7E+02	3.1E-03
100000	3.4E-04	2.0E-10	2.4E-11	7.1E-07	1.1E+02	1.4E+02	6.2E-03

TABLE 2. Computational results for Example 1, with $\varepsilon = 10^{-10}$.

N	t_{pre}/N	$E_{\max}$	E_{rms}	t_{eval}/N	M_{keep}/N	M_{pre}/N	t_{uncomp}/N
1000	2.2E-04	1.5E-10	8.9E-13	6.0E-07	1.1E+02	1.3E+03	4.0E-05
2500	2.9E-04	1.6E-09	3.6E-12	7.6E-07	1.1E+02	8.0E+02	9.2E-05
5000	2.6E-04	1.5E-09	1.6E-12	7.6E-07	1.2E+02	5.4E+02	1.8E-04
10000	2.9E-04	2.0E-09	1.0E-12	7.5E-07	1.2E+02	3.7E+02	3.6E-04
25000	3.9E-04	4.6E-09	1.4E-12	7.7E-07	1.1E+02	2.4E+02	9.1E-04
50000	4.3E-04	2.4E-09	5.5E-13	7.7E-07	1.1E+02	1.8E+02	1.9E-03
100000	5.5E-04	6.4E-08	2.5E-12	7.7E-07	1.1E+02	1.5E+02	3.7E-03

TABLE 3. Computational results for Example 2, with $\varepsilon = 10^{-10}$.

N	t_{pre}/N	$E_{\max}$	E_{rms}	t_{eval}/N	M_{keep}/N	M_{pre}/N	t_{uncomp}/N
1000	7.0E-04	9.1E-10	1.6E-10	1.1E-06	1.9E+02	1.4E+03	9.0E-05
2500	8.3E-04	2.6E-09	1.9E-10	1.2E-06	1.9E+02	8.0E+02	2.2E-04
5000	1.1E-03	3.6E-09	3.7E-10	1.3E-06	2.0E+02	5.7E+02	4.3E-04
10000	1.2E-03	4.6E-09	2.0E-10	1.3E-06	2.0E+02	4.2E+02	8.6E-04
25000	1.4E-03	5.8E-09	4.6E-10	1.2E-06	2.0E+02	3.0E+02	2.2E-03
50000	1.4E-03	4.7E-09	1.2E-10	1.2E-06	2.0E+02	2.6E+02	4.3E-03
100000	1.7E-03	2.2E-09	6.8E-11	1.2E-06	2.0E+02	2.3E+02	8.7E-03

TABLE 4. Computational results for Example 3, with $\varepsilon = 10^{-10}$.

REFERENCES

[1] J. Barnes and P. Hut. A hierarchical $o(n \log n)$ force calculation algorithm. *Nature*, 324, 1986.

[2] John P. Boyd. A fast algorithm for Chebyshev, Fourier, and sinc interpolation onto an irregular grid. *J. Comput. Phys.*, 103(2):243–257, 1992.

[3] H. Cheng, W. Crutchfield, Z. Gimbutas, L. Greengard, F. Ethridge, V. Rokhlin, N. Yarvin, and J. Zhao. A wideband fast multipole method for the Helmholtz equation in three dimensions. Technical Report 514, MadMax Optics, Inc., 2005.

[4] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.*, 26(4):1389–1404 (electronic), 2005.

[5] W. C. Chew, J. M. Jin, E. Michielsen, and J. Song. *Fast and Efficient Algorithms in Computational Electromagnetics.* Artech House, 2001.

	$\varepsilon = 10^{-3.5}$		$\varepsilon = 10^{-7}$		$\varepsilon = 10^{-14}$	
N	E_{rms}	t_{eval}/N	E_{rms}	t_{eval}/N	E_{rms}	t_{eval}/N
1000	2.1E-04	4.0E-07	5.7E-08	4.0E-07	5.1E-15	7.0E-07
2500	2.4E-04	4.8E-07	6.2E-08	5.6E-07	3.1E-15	8.4E-07
5000	2.3E-04	5.0E-07	8.1E-08	6.0E-07	3.8E-15	8.6E-07
10000	2.2E-04	4.9E-07	1.0E-07	6.1E-07	4.5E-15	8.5E-07
25000	2,2E-04	5.0E-07	1.1E-07	6.0E-07	7.0E-15	8.6E-07
50000	2.1E-04	5.1E-07	1.2E-07	6.1E-07	9.0E-15	8.6E-07
100000	2.0E-04	5.1E-07	1.3E-07	6.1E-07	1.1E-14	8.5E-07

TABLE 5. Computational results for Example 1 with three different values of ε.

N	1000	2500	5000	10000	25000	50000	100000
t_{fft}/N	5.0E-8	8.0E-8	6.0E-8	6.5E-8	8.4E-8	1.4E-7	1.6E-7
t_{matvec}/N	1.2E-5	3.2E-5	6.4E-5	—	—	—	—

TABLE 6. CPU times for the FFT and an uncompressed matrix-vector multiply with a stored matrix.

[6] Lokenath Debnath. *Integral transforms and their applications.* CRC Press, Boca Raton, FL, 1995.

[7] Zydrunas Gimbutas and Vladimir Rokhlin. A generalized fast multipole method for nonoscillatory kernels. *SIAM J. Sci. Comput.*, 24(3):796–817 (electronic), 2002.

[8] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[9] Leslie Greengard. *The rapid evaluation of potential fields in particle systems.* MIT Press, Cambridge, MA, 1988.

[10] Leslie Greengard and Vladimir Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. In *Acta numerica, 1997*, volume 6 of *Acta Numer.*, pages 229–269. Cambridge Univ. Press, Cambridge, 1997.

[11] Ming Gu and Stanley C. Eisenstat. A divide-and-conquer algorithm for the bidiagonal SVD. *SIAM J. Matrix Anal. Appl.*, 16(1):79–92, 1995.

[12] Ming Gu and Stanley C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Sci. Comput.*, 17(4):848–869, 1996.

[13] R. Jakob-Chien and B. Alpert. A fast spherical filter with uniform resolution. *J. Comput. Phys.*, 136:580–584, 1997.

[14] S. Kapur and D. Long. IES^3: Efficient electrostatic and electromagnetic simulation. *IEEE Comput. Sci. and Engineering*, 5(4):60–67, 1998.

[15] P. G. Martinsson and V. Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *J. Comput. Phys.*, 205(1):1–23, 2005.

[16] P.G. Martinsson, V. Rokhlin, and M. Tygert. On interpolation and integration in finite-dimensional spaces of bounded functions. Technical Report YALEU/DCS/RR-1317, Dept. of Computer Science, Yale, 2005.

[17] E. Michielssen and A. Boag. A multilevel matrix decomposition algorithm for analyzing scattering from large structures. *IEEE Trans. Antennas and Propagation*, 44:1086–1093, 1996.

[18] E. Michielssen, A. Boag, and W. C. Chew. Scattering from elongated objects: direct solution in $O(N \log^2 N)$ operations. *IEE Proc. Microw. Antennas Propag.*, 143(4):277 – 283, 1996.

[19] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White. Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. *SIAM J. Sci. Comput.*, 15(3):713–735, 1994. Iterative methods in numerical linear algebra (Copper Mountain Resort, CO, 1992).

[20] V. Rokhlin and M. Tygert. Fast algorithms for spherical harmonic expansions. *SIAM J. Sci. Comput.*, 27:1903 – 1928, 2006.

[21] D. Slepian and H.O. Pollak. Prolate spheroidal wave functions, fourier analysis, and uncertainty - i. *The Bell System Technical Journal*, pages 43–63, 1961.

[22] Reiji Suda and Shingo Kuriyama. Another preprocessing algorithm for generalized one-dimensional fast multipole method. *J. Comput. Phys.*, 195(2):790–803, 2004.

[23] Norman Yarvin and Vladimir Rokhlin. An improved fast multipole algorithm for potential fields on the line. *SIAM J. Numer. Anal.*, 36(2):629–666 (electronic), 1999.