SIAM J. SCI. COMPUT.
Vol. 34, No. 5, pp. A2507–A2532

A FAST DIRECT SOLVER FOR STRUCTURED LINEAR SYSTEMS BY RECURSIVE SKELETONIZATION*

KENNETH L. HO† AND LESLIE GREENGARD‡

Abstract. We present a fast direct solver for structured linear systems based on multilevel matrix compression. Using the recently developed interpolative decomposition of a low-rank matrix in a recursive manner, we embed an approximation of the original matrix into a larger but highly structured sparse one that allows fast factorization and application of the inverse. The algorithm extends the Martinsson–Rokhlin method developed for 2D boundary integral equations and proceeds in two phases: a precomputation phase, consisting of matrix compression and factorization, followed by a solution phase to apply the matrix inverse. For boundary integral equations which are not too oscillatory, e.g., based on the Green functions for the Laplace or low-frequency Helmholtz equations, both phases typically have complexity $\mathcal{O}(N)$ in two dimensions, where N is the number of discretization points. In our current implementation, the corresponding costs in three dimensions are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for precomputation and solution, respectively. Extensive numerical experiments show a speedup of $\sim$100 for the solution phase over modern fast multipole methods; however, the cost of precomputation remains high. Thus, the solver is particularly suited to problems where large numbers of iterations would be required. Such is the case with ill-conditioned linear systems or when the same system is to be solved with multiple right-hand sides. Our algorithm is implemented in Fortran and freely available.

Key words. fast algorithms, multilevel matrix compression, interpolative decomposition, sparse direct solver, integral equations, fast multipole method

AMS subject classifications. 65F05, 65F50, 65R20, 65Y15, 65Z05

DOI. 10.1137/120866683

1. Introduction.

Many problems in computational science and engineering require the solution of large, dense linear systems. Standard direct methods based on Gaussian elimination, of course, require $\mathcal{O}(N^3)$ work, where N is the system size. This quickly becomes infeasible as N increases. As a result, such systems are typically solved iteratively, combining GMRES [41], Bi-CGSTAB [44], or some other iterative scheme with fast algorithms to apply the system matrix, when available. For the integral equations of classical physics, this combination has led to some of the fastest solvers known today, with dramatically lower complexity estimates of the order $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ ([11, 33, 38] and references therein).

Despite their tremendous success, however, iterative methods still have several significant disadvantages when compared with their direct counterparts:

(i) *The number of iterations required by an iterative solver is highly sensitive to the conditioning of the system matrix.* Ill-conditioning arises, for example, in the solution of problems near resonance (particularly in the high-frequency regime), in geometries with "close-to-touching" interactions, in multicomponent physics models

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 21, 2012; accepted for publication (in revised form) July 17, 2012; published electronically September 13, 2012. This work was supported in part by the National Science Foundation under grant DGE-0333389, by the U.S. Department of Energy under contract DEFG0288ER25053, and by the Air Force Office of Scientific Research under NSSEFF program award FA9550-10-1-0180.
http://www.siam.org/journals/sisc/34-5/86668.html

†Courant Institute of Mathematical Sciences and Program in Computational Biology, New York University, New York, NY (ho@courant.nyu.edu).

‡Courant Institute of Mathematical Sciences, New York University, New York, NY (greengard@courant.nyu.edu).

with large contrasts in material properties, etc. Under these circumstances, the solution time can be far greater than expected. Direct methods, by contrast, are robust in the sense that their solution time does not degrade with conditioning. Thus, they are often preferred in production environments, where reliability of the solver and predictability of the solution time are important.

(ii) *One often wishes to solve a linear system governed by a fixed matrix with multiple right-hand sides.* This occurs, for example, in scattering problems, in optimization, and in the modeling of time-dependent processes in fixed geometry. Most iterative methods are unable to effectively exploit the fact that the system matrix is the same and simply treat each right-hand side as a new problem. Direct methods, on the other hand, are extremely efficient in this regard: once the system matrix has been factored, the matrix inverse can be applied to each right-hand side at a much lower cost.

(iii) *One often wishes to solve problems when the system matrix is altered by a low-rank modification.* Standard iterative methods do a poor job of exploiting this fact. Direct methods, on the other hand, can update the factorization of the original matrix using the Sherman–Morrison–Woodbury formula [29] or use the existing factorization as a preconditioner.

In this paper, we present an algorithm for the solution of structured linear systems that overcomes these deficiencies while remaining competitive with modern fast iterative solvers in many practical situations. The algorithm directly constructs a compressed ("data-sparse") representation of the system matrix inverse, assuming only that the matrix has a block low-rank structure similar to that utilized by fast matrix-vector product techniques like the fast multipole method (FMM) [22, 23]. Such matrices typically arise from the discretization of integral equations, where the low-rank structure can be understood in terms of far-field interactions between clusters of points, but the procedure is general and makes no a priori assumptions about rank. Our scheme is a multilevel extension of the work described in [21], which itself is based on the fast direct multilevel method developed for 2D boundary integral equations by Martinsson and Rokhlin [35].

While we do not seek to review the literature on fast direct solvers here, it is worth noting that similar efforts have been (and continue to be) pursued by various groups, most notably in the context of hierarchically semiseparable (HSS) matrices [6, 7, 49] and $\mathcal{H}$-matrices [26, 27, 28]. A short historical discussion can be found in [21], as well as in the recent article by Gillman, Young, and Martinsson [17]. The latter paper makes several improvements on the algorithm of [35] and presents a simple framework for understanding, implementing, and analyzing schemes for inverting integral equations on curves (that is, domains parametrized by a single variable). Planar domains with corners were treated recently in [4]. Applications to electromagnetic wave problems were considered in [45, 47]. Finally, it should be noted that Gillman's dissertation [16] includes 3D experiments that also extend the Martinsson–Rokhlin formalism to the case of integral equations on surfaces.

The present paper provides a mix of analysis, algorithmic work, and applications. The novelty of our contribution lies

(i) in the use of compression and auxilliary variables to embed an approximation of the original dense matrix into a sparse matrix framework that can make use of standard and well-developed sparse matrix technology;

(ii) in providing detailed numerical experiments in both 2D and 3D; and

(iii) in demonstrating the utility of fast direct solvers in several applications.

We believe that the scheme is substantially simpler to implement than prior schemes and that it leads to a more stable solution process.

As in previous schemes (see, e.g., [17]), the core algorithm in our work computes a compressed matrix representation using the interpolative decomposition (ID) [10, 32, 48] via a multilevel procedure that we refer to as *recursive skeletonization*. Once obtained, the compressed representation serves as a platform for fast matrix algebra including matrix-vector multiplication and matrix inversion. In its former capacity, the algorithm may be viewed as a generalized or kernel-independent FMM [19, 36, 50]; we explore this application in section 6. For matrix inversion, we show how to embed the compressed representation in an equivalent (but larger) sparse system, much in the style of [6, 39]. We then use a state-of-the-art sparse matrix solver to do the rest. We are grateful to David Bindel for initially suggesting an investigation of the sparse matrix formalism and rely in this paper on the sparse direct solver software UMFPACK [12, 13]. As in dense LU factorization, the direct solver is a two-phase process. First, following the generation of the compressed matrix embedding, a factored representation of the inverse is constructed. Second, in the solution phase, the matrix inverse is applied in a rapid manner to a specified right-hand side. As expected, the solution phase is very inexpensive, often beating a single FMM call by several orders of magnitude. For boundary integral equations without highly oscillatory kernels, e.g., the Green function for the Laplace or low-frequency Helmholtz equation, both phases typically have complexity $\mathcal{O}(N)$ in 2D. In 3D, the complexities in our current implementation are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for precomputation (compression and factorization) and solution, respectively.

The remainder of this paper is organized as follows. In section 2, we define the matrix structure of interest and review certain aspects of the ID. In section 3, we review the recursive skeletonization algorithm for matrix compression and describe the new formalism for embedding the compressed matrix in a sparse format. In section 4, we study the complexity of the algorithm for nonoscillatory problems, while in section 5, we give error estimates for applying a compressed matrix and its inverse. In section 6, we demonstrate the efficiency and generality of our scheme by reporting numerical results from its use as a generalized FMM, as a direct solver, and as an accelerator for molecular electrostatics and scattering problems. Finally, in section 7, we summarize our findings and discuss future work.

2. Preliminaries. In this section, we discuss the precise matrix structure that makes our fast solver possible. For this, let $A \in \mathbb{C}^{N \times N}$ be a matrix whose index vector $J = (1, 2, \ldots, N)$ is grouped into p contiguous blocks of n_i elements each, where $\sum_{i=1}^{p} n_i = N$:

$$J_i = \left(\sum_{j=1}^{i-1} n_j + 1, \sum_{j=1}^{i-1} n_j + 2, \ldots, \sum_{j=1}^{i} n_j \right), \quad i = 1, \ldots, p.$$

Then the linear system $A\mathbf{x} = \mathbf{b}$ can be written in the form

$$\sum_{j=1}^{p} A_{ij} \mathbf{x}_j = \mathbf{b}_i, \quad i = 1, \ldots, p,$$

where $\mathbf{x}_i, \mathbf{b}_i \in \mathbb{C}^{n_i}$, and $A_{ij} \in \mathbb{C}^{n_i \times n_j}$. Solution of the full linear system by classical Gaussian elimination is well known to require $\mathcal{O}(N^3)$ work.

DEFINITION 2.1 (block separability). *The matrix A is said to be* block separable *if each off-diagonal submatrix A_{ij} can be decomposed as the product of three low-rank matrices:*

$$A_{ij} = L_i S_{ij} R_j, \quad i \neq j, \tag{2.1}$$

where $L_i \in \mathbb{C}^{n_i \times k_i^{\mathrm{r}}}$, $S_{ij} \in \mathbb{C}^{k_i^{\mathrm{r}} \times k_j^{\mathrm{c}}}$, and $R_j \in \mathbb{C}^{k_j^{\mathrm{c}} \times n_j}$ with $k_i^{\mathrm{r}}, k_i^{\mathrm{c}} \ll n_i$. Note that in (2.1)*, the left matrix L_i depends only on the index i and the right matrix R_j depends only on the index j.*

We will see how such a factorization arises below. The term *block separable* was introduced in [17] and is closely related to that of semiseparable matrices [6, 7, 49] and $\mathcal{H}$-matrices [26, 27, 28]. In [21], the term *structured* was used, but block separable is somewhat more informative.

DEFINITION 2.2 (off-diagonal block rows and columns). *The i*th off-diagonal block row *of A is the submatrix $[\, A_{i1} \ \cdots \ A_{i(i-1)} \ A_{i(i+1)} \ \cdots \ A_{ip} \,]$ consisting of the ith block row of A with the diagonal block A_{ii} deleted; the* off-diagonal block columns *of A are defined analogously.*

Clearly, the block separability condition (2.1) is equivalent to requiring that the ith off-diagonal block row and column have rank k_i^{r} and k_i^{c}, respectively, for $i = 1, \ldots, p$ (see section 3 for details).

When A is block separable, it can be written as

$$A = D + LSR, \tag{2.2}$$

where

$$D = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{pp} \end{bmatrix} \in \mathbb{C}^{N \times N}$$

is block diagonal, consisting of the diagonal blocks of A,

$$L = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_p \end{bmatrix} \in \mathbb{C}^{N \times K_{\mathrm{r}}}, \qquad R = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_p \end{bmatrix} \in \mathbb{C}^{K_{\mathrm{c}} \times N}$$

are block diagonal, where $K_{\mathrm{r}} = \sum_{i=1}^{p} k_i^{\mathrm{r}}$ and $K_{\mathrm{c}} = \sum_{i=1}^{p} k_i^{\mathrm{c}}$, and

$$S = \begin{bmatrix} 0 & S_{12} & \cdots & S_{1p} \\ S_{21} & 0 & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \cdots & 0 \end{bmatrix} \in \mathbb{C}^{K_{\mathrm{r}} \times K_{\mathrm{c}}}$$

is dense with zero diagonal blocks. It is convenient to let $\mathbf{z} = R\mathbf{x}$ and $\mathbf{y} = S\mathbf{z}$. We can then write the original system in the form

$$\begin{bmatrix} D & L & \\ R & & -I \\ & -I & S \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{2.3}$$

This system is highly structured and sparse and can be efficiently factored using standard techniques. If we assume that each block corresponds to $N_i = N/p$ unknowns

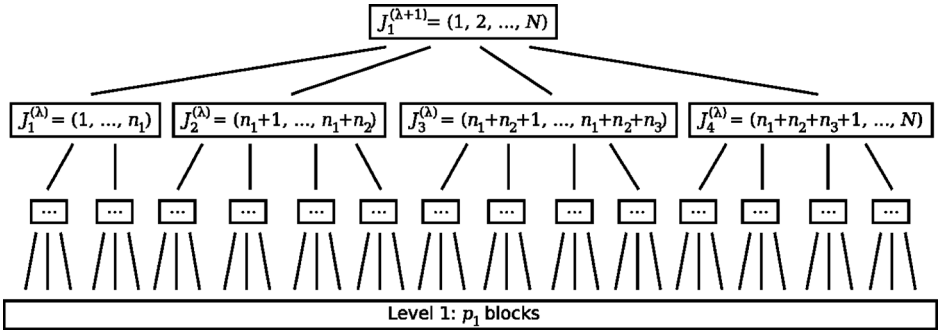

FIG. 2.1. *An example of a tree structure imposed on the index vector* $(1, 2, \ldots, N)$. *At each level of the hierarchy, a contiguous block of indices is divided into a set of children, each of which corresponds to a contiguous subset of indices.*

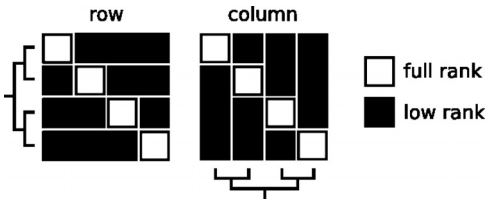

FIG. 2.2. *Matrix rank structure. At each level of the index tree, the off-diagonal block rows and columns (black) must have low numerical rank; the diagonal blocks (white) can in general be full-rank.*

and that the ranks $k_i^{\mathrm{r}} = k_i^{\mathrm{c}} \equiv k$ of the off-diagonal blocks are all the same, it is straightforward to see [17, 21] that a scheme based on (2.2) or (2.3) requires an amount of work of the order $\mathcal{O}(p(N/p)^3 + p^3k^3)$.

In many contexts (including integral equations), the notion of block separability is applicable on a hierarchy of subdivisions of the index vector. That is, a decomposition of the form (2.2) can be constructed at each level of the hierarchy. When a matrix has this structure, much more powerful solvers can be developed, but they will require some additional ideas (and notation).

2.1. Hierarchically structured matrices. Our treatment in this section follows that of [17]. Let $J = (1, 2, \ldots, N)$ be the index vector of a matrix $A \in \mathbb{C}^{N\times N}$. We assume that a tree structure τ is imposed on J which is $\lambda + 1$ levels deep. At level l, we assume that there are p_l nodes, with each such node $J_i^{(l)}$ corresponding to a contiguous subsequence of J such that

$$\left\{J_1^{(l)}, J_2^{(l)}, \ldots, J_{p_l}^{(l)}\right\} = J.$$

We denote the *finest level* as level 1 and the coarsest level as level $\lambda+1$ (which consists of a single block). Each node $J_i^{(l)}$ at level $l > 1$ has a finite number of children at level $l - 1$ whose concatenation yields the indices in $J_i^{(l)}$ (Figure 2.1).

The matrix A is *hierarchically block separable* [17] if it is block separable at each level of the hierarchy defined by τ. In other words, it is structured in the sense of the present paper if, on each level of τ, the off-diagonal block rows and columns are low-rank (Figure 2.2). Such matrices arise, for example, when discretizing integral equations with nonoscillatory kernels (up to a specified precision).

Example 1. Consider the integral operator

$$\phi(x) = \int G(x,y)\,\rho(y)\,dy, \tag{2.4}$$

where

$$G(x,y) = -\frac{1}{2\pi}\log|x-y| \tag{2.5}$$

is the Green function for the 2D Laplace equation, and the domain of integration is a square B in the plane. This is a 2D *volume integral operator*. Suppose now that we discretize (2.4) on a $\sqrt{N}\times\sqrt{N}$ grid:

$$\phi(x_i) = \frac{1}{N}\sum_{j\neq i} G(x_i,x_j)\,\rho(x_j). \tag{2.6}$$

(This is not a high-order quadrature, but that is a separate issue.) Let us superimpose on B a quadtree of depth $\lambda+1$, where B is the root node (level $\lambda+1$). Level λ is obtained from level $\lambda+1$ by subdividing the box B into four equal squares and reordering the points x_i so that each child holds a contiguous set of indices. This procedure is carried out until level 1 is reached, reordering the nodes at each level so that the points contained in every node at every level correspond to a contiguous set of indices. It is clear that with this ordering, the matrix corresponding to (2.6) is hierarchically block separable, since the interactions between nonadjacent boxes at every level are low-rank to any specified precision (from standard multipole estimates [22]). Adjacent boxes are low-rank for a more subtle reason (see section 4 and Figure 4.1).

Example 2. Suppose now that we wish to solve an interior Dirichlet problem for the Laplace equation in a simply connected 3D domain Ω with boundary $\partial\Omega$:

$$\Delta u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \partial\Omega. \tag{2.7}$$

Potential theory [24] suggests that we seek a solution in the form of a double-layer potential

$$u(x) = \int_{\partial\Omega} \frac{\partial G}{\partial \nu_y}(x,y)\,\sigma(y)\,dy \quad \text{for } x\in\Omega, \tag{2.8}$$

where

$$G(x,y) = \frac{1}{4\pi|x-y|} \tag{2.9}$$

is the Green function for the 3D Laplace equation, ν_y is the unit outer normal at $y\in\partial\Omega$, and σ is an unknown surface density. Letting x approach the boundary, this gives rise to the second-kind Fredholm equation

$$-\frac{1}{2}\sigma(x) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_y}(x,y)\,\sigma(y)\,dy = f(x). \tag{2.10}$$

Using a standard collocation scheme based on piecewise constant densities over a triangulated surface, we enclose $\partial\Omega$ in a box B and bin sort the triangle centroids using an octree where, as in the previous example, we reorder the nodes so that each box in the hierarchy contains contiguous indices. It can be shown that the resulting matrix is also hierarchically block separable (see section 4 and [21]).

We turn now to a discussion of the ID, the compression algorithm that we will use to compute low-rank approximations of off-diagonal blocks. A useful feature of the ID is that it is able to compute the rank of a matrix on the fly, since the exact ranks of the blocks are difficult to ascertain a priori—that is, the ID is *rank-revealing*.

2.2. ID. Many decompositions exist for low-rank matrix approximation, including the singular value decomposition, which is well-known to be optimal [20]. Here, we consider instead the ID [10, 32, 48], which produces a near-optimal representation that is more useful for our purposes as it permits an efficient scheme for multilevel compression when used in a hierarchical setting.

DEFINITION 2.3 (ID). *Let $A \in \mathbb{C}^{m\times n}$ be a matrix and $\|\cdot\|$ the matrix 2-norm. A rank-k approximation of A in the form of an* ID *is a representation $A \approx BP$, where $B \in \mathbb{C}^{m\times k}$, whose columns constitute a subset of the columns of A, and $P \in \mathbb{C}^{k\times n}$, a subset of whose columns makes up the $k \times k$ identity matrix, such that $\|P\|$ is small and $\|A - BP\| \sim \sigma_{k+1}$, where σ_{k+1} is the $(k+1)$st greatest singular value of A. We call B and P the* skeleton *and* projection matrices*, respectively.*

Clearly, the ID compresses the column space of A; to compress the row space, simply apply the ID to A^{T}, which produces an analogous representation $A = \tilde{P}\tilde{B}$, where $\tilde{P} \in \mathbb{C}^{m\times k}$ and $\tilde{B} \in \mathbb{C}^{k\times n}$.

DEFINITION 2.4 (row and column skeletons). *The row indices that corrrespond to the retained rows in the ID are called the* row *or* incoming skeletons*. The column indices that corrrespond to the retained columns in the ID are called the* column *or* outgoing skeletons*.*

Reasonably efficient schemes for constructing an ID exist [10, 32, 48]. By combining such schemes with methods for estimating the approximation error, we can compute an ID to any relative precision $\epsilon > 0$ by adaptively determining the required rank k [32]. This is the sense in which we will use the ID.

While previous related work [21, 35] used the deterministic $\mathcal{O}(kmn)$ algorithm of [10], we employ here the latest compression technology based on random sampling, which typically requires only $\mathcal{O}(mn \log k + k^2 n)$ operations [32, 48].

3. Algorithm.

In this section, we first describe the "standard" ID-based fast multilevel matrix compression algorithm (as in [17, 35]). The HSS and $\mathcal{H}$-matrix formalisms use the same underlying philosophy [6, 7, 26, 27, 28, 49]. We then describe our new inversion scheme.

3.1. Hierarchical matrix compression. Let $A \in \mathbb{C}^{N\times N}$ be a matrix with $p \times p$ blocks, structured in the sense of section 2.1, and $\epsilon > 0$ a target relative precision. We first outline a one-level matrix compression scheme:

1. For $i = 1, \ldots, p$, use the ID to compress the row space of the ith off-diagonal block row to precision ϵ. Let L_i denote the corresponding row projection matrix.

2. Similarly, for $j = 1, \ldots, p$, use the ID to compress the column space of the jth off-diagonal block column to precision ϵ. Let R_j denote the corresponding column projection matrix.

3. Approximate the off-diagonal blocks of A by $A_{ij} \approx L_i S_{ij} R_j$ for $i \neq j$, where S_{ij} is the submatrix of A_{ij} defined by the row and column skeletons associated with L_i and R_j, respectively.

This yields precisely the matrix structure discussed in section 2, following (2.1). The one-level scheme is illustrated graphically in Figure 3.1.

The multilevel algorithm is now just a simple extension based on the observation that by ascending one level in the index tree and regrouping blocks accordingly, we can compress the skeleton matrix S in (2.2) in exactly the same form, leading to a procedure that we naturally call *recursive skeletonization* (Figure 3.2).

The full algorithm may be specified as follows:

1. Starting at the leaves of the tree, extract the diagonal blocks and perform one level of compression of the off-diagonal blocks.

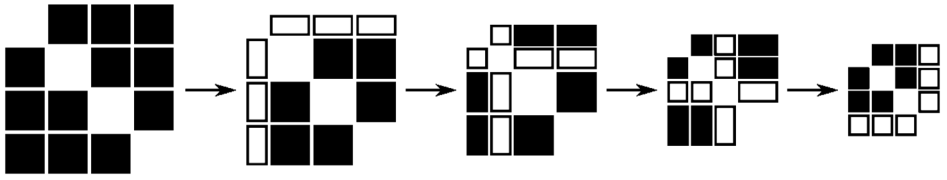

FIG. 3.1. *One level of matrix compression, obtained by sequentially compressing the off-diagonal block rows and columns. At each step, the matrix blocks whose row or column spaces are being compressed are highlighted in white.*

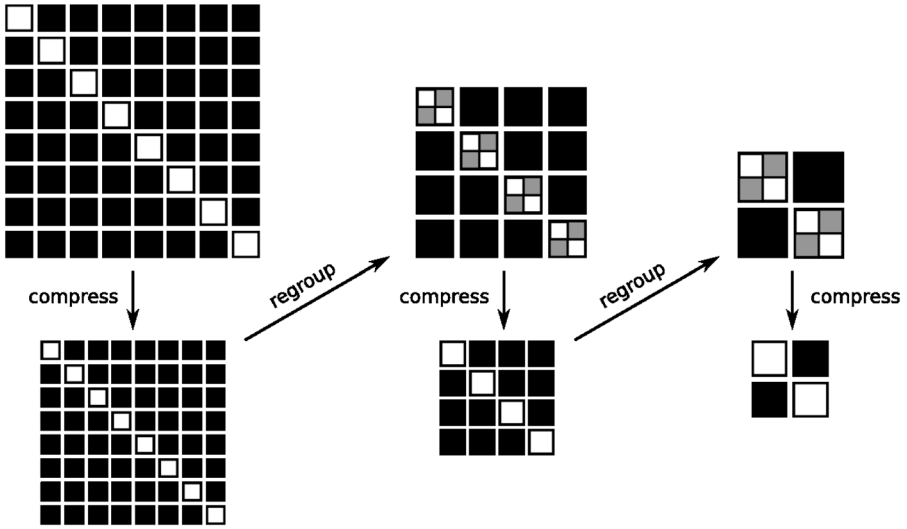

FIG. 3.2. *Multilevel matrix compression, comprising alternating steps of compression and regrouping via ascension of the index tree. The diagonal blocks (white and gray) are not compressed but are instead extracted at each level of the tree; they are shown here only to illustrate the regrouping process.*

2. Move up one level in the tree and regroup the matrix blocks according to the tree structure. Terminate if the new level is the root; otherwise, extract the diagonal blocks, recompress the off-diagonal blocks, and repeat.

The result is a telescoping representation

$$A \approx D^{(1)} + L^{(1)} \left[D^{(2)} + L^{(2)} \left(\cdots D^{(\lambda)} + L^{(\lambda)} S R^{(\lambda)} \cdots \right) R^{(2)} \right] R^{(1)}, \tag{3.1}$$

where the superscript indexes the compression level $l = 1, \ldots, \lambda$.

Example 3. As a demonstration of the multilevel compression technique, consider the matrix defined by $N = 8192$ points uniformly distributed in the unit square, interacting via the 2D Laplace Green's function (2.5) and sorted according to a quadtree ordering. The sequence of skeletons remaining after each level of compression to $\epsilon = 10^{-3}$ is shown in Figure 3.3, from which we see that compression creates a *sparsification* of the sources which, in a geometric setting, leaves skeletons along the boundaries of each block.

3.2. Accelerated compression via proxy surfaces. The computational cost of the algorithm described in the previous section is dominated by the fact that each

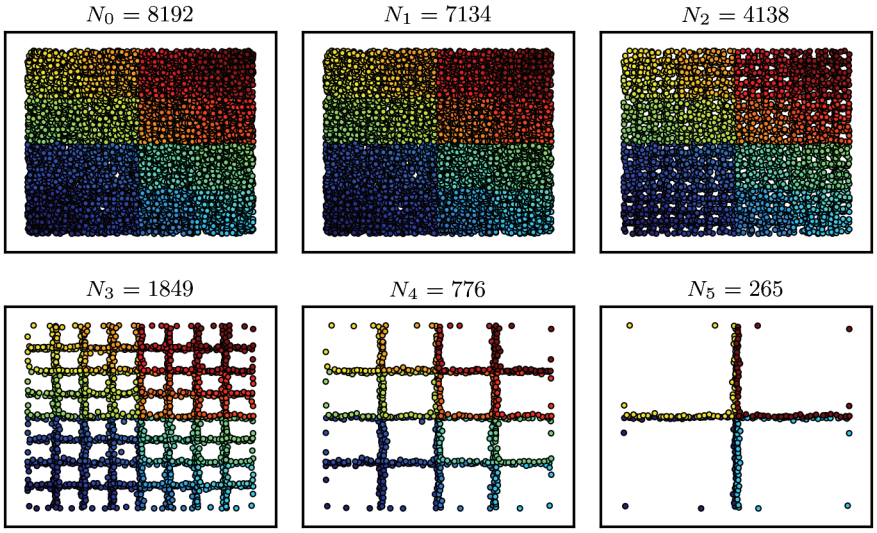

FIG. 3.3. *Sparsification by recursive skeletonization. Logarithmic interactions between* $N = 8192$ *points in the unit square are compressed to relative precision* $\epsilon = 10^{-3}$ *using a five-level quadtree-based scheme. At each level, the surviving skeletons are shown, colored by block index, with the total number of skeletons remaining given by* N_l *for compression level* $l = 0, \ldots, 5$*, where* $l = 0$ *denotes the original uncompressed system.*

step is global: that is, compressing the row or column space for each block requires accessing all other blocks in that row or column. If no further knowledge of the matrix is available, this is indeed necessary. However, as noted by [10, 21, 35, 37], this global work can often be replaced by a local one, resulting in considerable savings.

A sufficient condition for this acceleration is that the matrix correspond to evaluating a potential field for which some form of Green's identities hold. It is easiest to present the main ideas in the context of Laplace's equation. For this, consider Figure 3.4, which depicts a set of sources in the plane. We assume that block index i corresponds to the sources in the central square B. The ith off-diagonal block row then corresponds to the interactions of all points outside B with all points inside B. We can separate this into contributions from the near neighbors of B, which are local, and the distant sources, which lie outside the near-neighbor domain, whose boundary is denoted by Γ. But any field induced by the distant sources induces a harmonic function inside Γ and can therefore be replicated by a charge density on Γ itself. Thus, rather than using the detailed structure of the distant points, the row (incoming) skeletons for B can be extracted by considering just the combination of the near-neighbor sources and an artifical set of charges placed on Γ, which we refer to as a *proxy surface*. Likewise, the column (outgoing) skeletons for B can be determined by considering only the near neighbors and the proxy surface. If the potential field is correct on the proxy surface, it will be correct at all more distant points (again via some variant of Green's theorem).

The interaction rank between Γ and B is constant (depending on the desired precision) from standard multipole estimates [22, 23]. In summary, the number of points required to discretize Γ is constant, and the dimension of the matrix to compress against for the block corresponding to B is essentially just the number of points in the physically neighboring blocks.

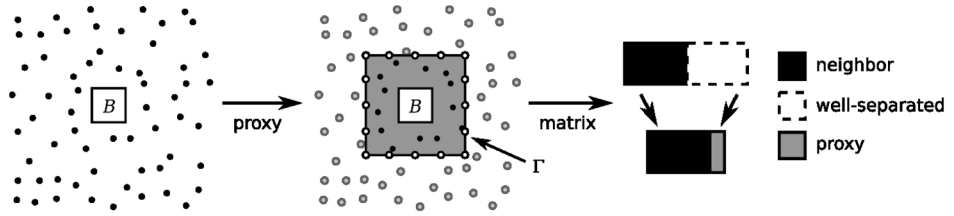

FIG. 3.4. *Accelerated compression using proxy surfaces. The field within a region B due to a distribution of exterior sources (left) can be decomposed into neighboring and well-separated contributions. By representing the latter via a proxy surface Γ (center), the matrix dimension to compress against for the block corresponding to B (right) can be reduced to the number of neighboring points plus a constant set of points on Γ, regardless of how many points lie beyond Γ.*

Similar arguments hold for other kernels of potential theory including the heat, Helmholtz, Yukawa, Stokes, and elasticity kernels, though care must be taken for oscillatory problems which could require a combination of single and double layer potentials to avoid spurious resonances in the representation for the exterior.

3.3. Compressed matrix-vector multiplication. The compressed representation (3.1) admits an obvious fast algorithm for computing the matrix-vector product $\mathbf{y} = A\mathbf{x}$. As shown in [17], one simply applies the matrices in (3.1) from right to left. Like the FMM, this procedure can be thought of as occurring in two passes:

1. an upward pass, corresponding to the sequential application of the column projection matrices $R^{(l)}$, which hierarchically compress the input data $\mathbf{x}$ to the column (outgoing) skeleton subspace;

2. a downward pass, corresponding to the sequential application of the row projection matrices $L^{(l)}$, which hierarchically project onto the row (incoming) skeleton subspace and, from there, back onto the output elements $\mathbf{y}$.

3.4. Compressed matrix inversion. The representation (3.1) also permits a fast algorithm for the direct inversion of nonsingular matrices. The one-level scheme was discussed in section 2. In the multilevel scheme, the system $S\mathbf{z} = \mathbf{y}$ in (2.3) is itself expanded in the same form, leading to the sparse embedding

$$
\begin{bmatrix}
D^{(1)} & L^{(1)} & & & & & & \\
R^{(1)} & & -I & & & & & \\
 & -I & D^{(2)} & L^{(2)} & & & & \\
 & & R^{(2)} & \ddots & \ddots & & & \\
 & & & \ddots & D^{(\lambda)} & L^{(\lambda)} & & \\
 & & & & R^{(\lambda)} & & -I & \\
 & & & & & -I & S &
\end{bmatrix}
\begin{bmatrix}
\mathbf{x} \\ \mathbf{y}^{(1)} \\ \mathbf{z}^{(1)} \\ \vdots \\ \vdots \\ \mathbf{y}^{(\lambda)} \\ \mathbf{z}^{(\lambda)}
\end{bmatrix}
=
\begin{bmatrix}
\mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \vdots \\ \mathbf{0} \\ \mathbf{0}
\end{bmatrix}. \tag{3.2}
$$

To understand the consequences of this sparse representation, it is instructive to consider the special case in which the row and column skeleton dimensions are identical for each block, say, k, so that the total row and column skeleton dimensions are $K \equiv K_{\mathrm{r}} = K_{\mathrm{c}} = pk$. Then, studying (2.3) first and assuming that D is invertible, block elimination of $\mathbf{x}$ and $\mathbf{y}$ yields

$$(\Lambda + S)\,\mathbf{z} = \Lambda R D^{-1}\mathbf{b},$$

where $\Lambda = (RD^{-1}L)^{-1} \in \mathbb{C}^{K\times K}$ is block diagonal. Back substitution then yields

$$\mathbf{x} = \left[D^{-1} - D^{-1}L\Lambda RD^{-1} + D^{-1}L\Lambda\left(\Lambda + S\right)^{-1}\Lambda RD^{-1}\right]\mathbf{b}.$$

In other words, the matrix inverse is

$$A^{-1} \approx \mathcal{D} + \mathcal{L}\mathcal{S}^{-1}\mathcal{R}, \tag{3.3}$$

where

$$\mathcal{D} = D^{-1} - D^{-1}L\Lambda RD^{-1} \in \mathbb{C}^{N\times N}$$

and

$$\mathcal{L} = D^{-1}L\Lambda \in \mathbb{C}^{N\times K}, \qquad \mathcal{R} = \Lambda RD^{-1} \in \mathbb{C}^{K\times N}$$

are all block diagonal, and

$$\mathcal{S} = \Lambda + S \in \mathbb{C}^{K\times K}$$

is dense. Note that $\mathcal{S}$ is equal to the skeleton matrix S with its diagonal blocks filled in. Thus, (3.3) is a compressed representation of A^{-1} with minimal fill-in over the original compressed representation (2.2) of A. In the multilevel setting, one carries out the above factorization recursively, since $\mathcal{S}$ can now be inverted in the same manner:

$$A^{-1} \approx \mathcal{D}^{(1)} + \mathcal{L}^{(1)}\left[\mathcal{D}^{(2)} + \mathcal{L}^{(2)}\left(\cdots\mathcal{D}^{(\lambda)} + \mathcal{L}^{(\lambda)}\mathcal{S}^{-1}\mathcal{R}^{(\lambda)}\cdots\right)\mathcal{R}^{(2)}\right]\mathcal{R}^{(1)}. \tag{3.4}$$

This point of view is elaborated in [17].

In the general case, the preceding procedure may fail if D happens to be singular and (more generally) may be numerically unstable if care is not taken to stabilize the block elimination scheme using some sort of pivoting. Thus, rather than using the "hand-rolled" Gaussian elimination scheme of [16, 17, 35] to compute the telescoping inverse (3.4), we rely instead on the existence of high-quality sparse direct solver software. More precisely, we simply supply UMFPACK with the sparse representation (3.2) and let it compute the corresponding factorization. Numerical results show that the performance is similar to that expected from (3.4).

4. Complexity analysis. For the sake of completeness, we briefly analyze the complexity of the algorithm presented in section 3 for a typical example: discretization of the integral operator (2.4), where the integral kernel has smoothness properties similar to that of the Green function for the Laplace equation. We follow the analysis of [16, 17, 35] and estimate costs for the hand-rolled Gaussian elimination scheme. We ignore quadrature issues and assume that we are given a matrix A acting on N points distributed randomly in a d-dimensional domain, sorted by an orthtree that uniformly subdivides until all block sizes are $\mathcal{O}(1)$. (In 1D, an orthtree is a binary tree; in 2D, it is a quadtree; and in 3D, it is an octree.)

For each compression level $l = 1, \ldots, \lambda$, with $l = 1$ being the finest, let p_l be the number of matrix blocks and n_l and k_l the uncompressed and compressed block dimensions, respectively, assumed equal for all blocks and identical across rows and columns, for simplicity. We first make the following observations:

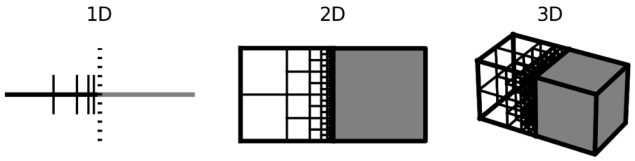

FIG. 4.1. *The interaction rank between two adjacent blocks can be calculated by recursively subdividing the source block (white) into well-separated subblocks with respect to the target (gray), each of which has constant rank.*

(i) The total matrix dimension is $p_1 n_1 = N$, where $n_1 = \mathcal{O}(1)$, so $p_1 \sim N$.

(ii) Each subdivision increases the number of blocks by a factor of roughly 2^d, so $p_l \sim p_{l-1}/2^d \sim p_1/2^{d(l-1)}$. In particular, $p_\lambda = \mathcal{O}(1)$, so $\lambda \sim \log_{2^d} N = (1/d)\log N$.

(iii) The total number of points at level $l > 1$ is equal to the total number of skeletons at level $l-1$, i.e., $p_l n_l = p_{l-1} k_{l-1}$, so $n_l \sim 2^d k_{l-1}$.

Furthermore, we note that k_l is on the order of the interaction rank between two adjacent blocks at level l, which can be analyzed by recursive subdivision of the source block to expose well-separated structures with respect to the target (Figure 4.1). Assuming only that the interaction between a source subregion separated from a target by a distance of at least its own size is of constant rank (to a fixed precision ϵ), we have

$$k_l \sim \sum_{l=1}^{\log_{2^d} n_l} 2^{(d-1)l} \sim \begin{cases} \log n_l & \text{if } d = 1, \\ n_l^{1-1/d} & \text{if } d > 1, \end{cases}$$

where, clearly, $n_l \sim (p_1/p_l) n_1 \sim 2^{d(l-1)} n_1$, so

$$k_l \sim \begin{cases} (l-1)\log 2 + \log n_1 & \text{if } d = 1, \\ 2^{(d-1)(l-1)} n_1^{1-1/d} & \text{if } d > 1. \end{cases}$$

4.1. Matrix compression. From section 2.2, the cost of computing a rank-k ID of an $m \times n$ matrix is $\mathcal{O}(mn \log k + k^2 n)$. We will only consider the case of proxy compression, for which $m = \mathcal{O}(n_l)$ for a block at level l, so the total cost is

$$T_{\text{cm}} \sim \sum_{l=1}^{\lambda} p_l \left(n_l^2 \log k_l + k_l^2 n_l \right) \sim \begin{cases} N & \text{if } d = 1, \\ N^{3(1-1/d)} & \text{if } d > 1. \end{cases} \tag{4.1}$$

4.2. Matrix-vector multiplication. The cost of applying $D^{(l)}$ is $\mathcal{O}(p_l n_l^2)$, while that of applying $L^{(l)}$ or $R^{(l)}$ is $\mathcal{O}(p_l k_l n_l)$. Combined with the $\mathcal{O}((p_\lambda k_\lambda)^2)$ cost of applying S, the total cost is

$$T_{\text{mv}} \sim \sum_{l=1}^{\lambda} p_l n_l \left(k_l + n_l \right) + \left(p_\lambda k_\lambda \right)^2 \sim \begin{cases} N & \text{if } d = 1, \\ N \log N & \text{if } d = 2, \\ N^{2(1-1/d)} & \text{if } d > 2. \end{cases} \tag{4.2}$$

4.3. Matrix factorization and inverse application. We turn now to the analysis of the cost of factorization using (3.4). At each level 1, the cost of constructing $\mathcal{D}^{(1)}$, $\mathcal{L}^{(1)}$, and $\mathcal{R}^{(1)}$ is $\mathcal{O}(p_1 n_1^3)$; at the final level, the cost of constructing and inverting $\mathcal{S}$ is $\mathcal{O}((p_\lambda k_\lambda)^3)$. Thus, the total cost is

$$T_{\text{lu}} \sim \sum_{l=1}^{\lambda} p_l n_l^3 + \left(p_\lambda k_\lambda \right)^3,$$

which has complexity (4.1).

Finally, we note that the dimensions of $\mathcal{D}^{(l)}$, $\mathcal{L}^{(l)}$, $\mathcal{R}^{(l)}$, and $\mathcal{S}^{-1}$ are the same as those of $D^{(l)}$, $L^{(l)}$, $R^{(l)}$, and S, respectively. Thus, the total cost of applying the inverse, denoted by T_{sv}, has the same complexity as T_{mv}, namely, (4.2).

In our UMFPACK-based approach, the estimation of cost is a rather complicated task, and we do not attempt to carry out a detailed analysis of its performance. Suffice it to say, there is a one-to-one correspondence between the hand-rolled Gaussian elimination approach and one possible elimination scheme in UMFPACK. Since that solver is highly optimized, the asymptotic cost should be the same (or better). For some matrices, it is possible that straight Gaussian elimination may be unstable without pivoting, while UMFPACK will carry out a backward-stable scheme. This is a distinct advantage of the sparse matrix approach although the complexity and fill-in analysis then becomes more involved.

4.4. Storage. An important issue in direct solvers, of course, is that of storage requirements. In the present setting the relevant matrices are the compressed sparse representation (3.2) and the factorization computed within UMFPACK. This will be (4.2) for the forward operator and, in the absence of pivoting, for the sparse factorization as well. If pivoting is required, the analysis is more complex as it involves some matrix fill-in and is postponed to future work.

5. Error analysis. We now state some simple error estimates for applying and inverting a compressed matrix. Let A be the original matrix and A_ϵ its compressed representation, constructed using the algorithm of section 3 such that

$$\frac{\|A - A_\epsilon\|}{\|A\|} \leq \epsilon$$

for some $\epsilon > 0$. Note that this need not be the same as the specified local precision in the ID since errors may accumulate across levels. However, as in [17], we have found that such error propagation is mild.

Let $\mathbf{x}$ and $\mathbf{b}$ be vectors such that $A\mathbf{x} = \mathbf{b}$. Then it is straightforward to verify that for $\mathbf{b}_\epsilon = A_\epsilon \mathbf{x}$,

$$\frac{\|\mathbf{b} - \mathbf{b}_\epsilon\|}{\|\mathbf{b}\|} \leq \epsilon \|A\| \left\|A^{-1}\right\| = \epsilon \kappa(A),$$

where $\kappa(A)$ is the condition number of A. Furthermore, if $\mathbf{x}_\epsilon = A_\epsilon^{-1}\mathbf{b}$, then

$$\frac{\|\mathbf{x} - \mathbf{x}_\epsilon\|}{\|\mathbf{x}\|} \leq \frac{2\epsilon\kappa(A)}{1 - \epsilon\kappa(A)}.$$

In particular, if A is well-conditioned, e.g., A is the discretization of a second-kind integral equation, then $\kappa(A) = \mathcal{O}(1)$, so

$$\frac{\|\mathbf{x} - \mathbf{x}_\epsilon\|}{\|\mathbf{x}\|} = \mathcal{O}(\epsilon).$$

6. Numerical examples. In this section, we investigate the efficiency and flexibility of our algorithm by considering some representative examples. We begin with timing benchmarks for the Laplace and Helmholtz kernels in 2D and 3D, using the algorithm both as an FMM and as a direct solver, followed by applications in molecular electrostatics and multiple scattering.

All matrices were blocked using quadtrees in 2D and octrees in 3D, uniformly subdivided until all block sizes were $\mathcal{O}(1)$, but adaptively truncating empty boxes during the refinement process. Only proxy compression was considered, with proxy surfaces constructed on the boundary of the supercell enclosing the neighbors of each block. We discretized all proxy surfaces using a constant number of points, independent of the matrix size N: for the Laplace equation, this constant depended only on the compression precision ϵ, while for the Helmholtz equation, it depended also on the wave frequency, chosen to be consistent with the Nyquist–Shannon sampling theorem. Computations were performed over $\mathbb{R}$ instead of $\mathbb{C}$, where possible. The algorithm was implemented in Fortran, and all experiments were performed on a 2.66 GHz processor in double precision.

In many instances, we compare our results against those obtained using LAPACK/ATLAS [2, 46] and the FMM [9, 22, 23, 40]. All FMM timings were computed using the open-source FMMLIB package [18], which is a fairly efficient implementation but does not include the plane-wave optimizations of [9, 23] or the diagonal translation operators of [40].

6.1. Generalized FMM. We first considered the use of recursive skeletonization as a generalized FMM for the rapid computation of matrix-vector products.

6.1.1. The Laplace equation. We considered two point distributions in the plane: points on the unit circle and in the unit square, hereafter referred to as the 2D surface and volume cases, respectively. We assumed that the governing matrix corresponds to the interaction of charges via the Green's function (2.5). The surface case is typical of layer-potential evaluation when using boundary integral equations. Since a domain boundary in 2D can be described by a single parameter (such as arclength), it is a 1D domain, so the expected complexities from section 4 correspond to $d = 1$: $\mathcal{O}(N)$ work for both matrix compression and application. (See [17] for a detailed discussion of the $d = 1$ case.) In the volume case, the dimension is $d = 2$, so the expected complexities are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for compression and application, respectively.

For the 3D Laplace kernel (2.9), we considered surface and volume point geometries on the unit sphere and within the unit cube, respectively. The corresponding dimensions are $d = 2$ and $d = 3$. Thus, the expected complexities for the 3D surface case are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for compression and application, respectively, while those for the 3D volume case are $\mathcal{O}(N^2)$ and $\mathcal{O}(N^{4/3})$, respectively.

We present timing results for each case and compare with LAPACK/ATLAS and the FMM for a range of N at $\epsilon = 10^{-9}$. Detailed data are provided in Tables 6.1, 6.2, 6.3, and 6.4 and plotted in Figure 6.1. It is evident that our algorithm scales as predicted. Its performance in 2D is particularly strong. Not only does our algorithm beat the $\mathcal{O}(N^2)$ uncompressed matrix-vector product for modest N, it is faster even than the $\mathcal{O}(N)$ FMM (at least after compression). In 3D, the same is true over the range of N tested, although the increase in asymptotic complexity would eventually make the scheme less competitive. In all cases studied, the compression time T_{cm} was larger than the time to apply the FMM by one (2D surface) to two (all other cases) orders of magnitude, while the compressed matrix-vector product time T_{mv} was consistently smaller by the same amount. Thus, our algorithm also shows promise as a fast iterative solver for problems requiring more than $\sim$10–100 iterations. Furthermore, we note the effectiveness of compression: for $N = 131072$, the storage requirement for the uncompressed matrix is 137 GB, whereas that for the compressed representations are only 100 MB and 1.2 GB in the 2D surface and volume cases,

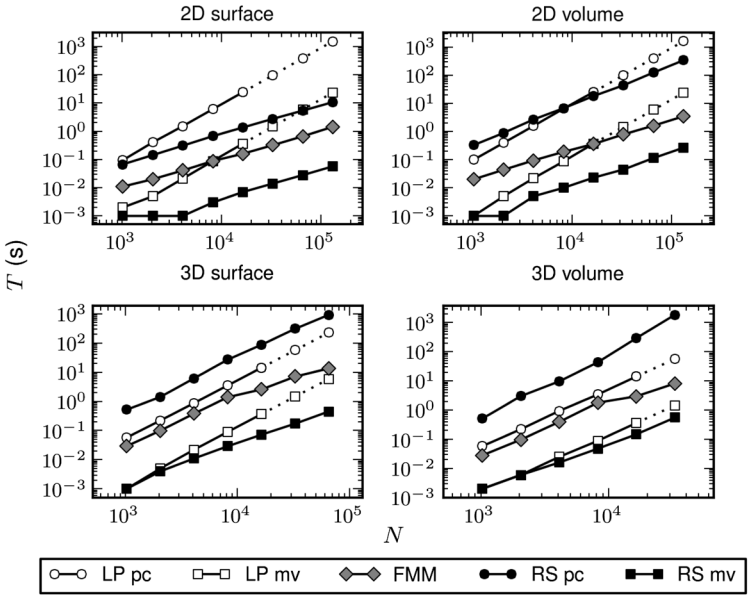

FIG. 6.1. *CPU times for applying the Laplace kernel in various cases using LAPACK/ATLAS (LP), the FMM, and recursive skeletonization (RS) as a function of the matrix size N. For LP and RS, the computation is split into two parts: precomputation (pc), for LP consisting of matrix formation and for RS of matrix compression, and matrix-vector multiplication (mv). The precision of the FMM and RS was set at $\epsilon = 10^{-9}$. Dotted lines indicate extrapolated data.*

TABLE 6.1

Numerical results for applying the Laplace kernel in the 2D surface case at precision $\epsilon = 10^{-9}$: N, uncompressed matrix dimension; K_r, row skeleton dimension; K_c, column skeleton dimension; T_{cm}, matrix compression time (s); T_{mv}, matrix-vector multiplication time (s); E, relative error; M, required storage for compressed matrix (MB).

N	K_r	K_c	T_{cm}	T_{mv}	E	M
1024	94	94	6.7E−2	1.0E−3	3.1E−8	8.5E−1
2048	105	104	1.4E−1	1.0E−3	4.5E−8	1.7E+0
4096	113	114	3.1E−1	1.0E−3	1.1E−7	3.4E+0
8192	123	123	6.7E−1	3.0E−3	4.4E−7	6.4E+0
16384	133	134	1.4E+0	7.0E−3	4.0E−7	1.3E+1
32768	142	142	2.7E+0	1.4E−2	4.7E−7	2.5E+1
65536	150	149	5.4E+0	2.8E−2	9.4E−7	5.0E+1
131072	159	158	1.1E+1	5.7E−2	9.8E−7	1.0E+2

TABLE 6.2

Numerical results for applying the Laplace kernel in the 2D volume case at precision $\epsilon = 10^{-9}$; notation is as in Table 6.1.

N	K_r	K_c	T_{cm}	T_{mv}	E	M
1024	299	298	3.3E−1	1.0E−3	3.6E−10	2.9E+0
2048	403	405	8.9E−1	1.0E−3	3.7E−10	7.1E+0
4096	570	570	2.7E+0	5.0E−3	1.0E−09	1.8E+1
8192	795	793	6.8E+0	1.0E−2	8.8E−10	4.3E+1
16384	1092	1091	1.8E+1	2.3E−2	7.7E−10	1.0E+2
32768	1506	1505	4.4E+1	4.5E−2	1.0E−09	2.3E+2
65536	2099	2101	1.3E+2	1.1E−1	1.1E−09	5.3E+2
131072	2904	2903	3.4E+2	2.7E−1	1.1E−09	1.2E+3

TABLE 6.3

Numerical results for applying the Laplace kernel in the 3D surface case at precision $\epsilon = 10^{-9}$; notation is as in Table 6.1.

N	K_r	K_c	T_{cm}	T_{mv}	E	M
1024	967	967	5.2E−1	1.0E−3	1.0E−11	7.7E+0
2048	1531	1532	1.4E+0	4.0E−3	1.8E−10	2.2E+1
4096	2298	2295	6.1E+0	1.1E−2	1.4E−10	6.2E+1
8192	3438	3426	2.7E+1	2.9E−2	1.2E−10	1.7E+2
16384	4962	4950	8.7E+1	7.2E−2	3.0E−10	4.2E+2
32768	6974	6987	3.1E+2	1.7E−1	4.3E−10	9.9E+2
65536	9899	9925	9.2E+2	4.5E−1	7.7E−10	2.3E+3

TABLE 6.4

Numerical results for applying the Laplace kernel in the 3D volume case at precision $\epsilon = 10^{-9}$; notation is as in Table 6.1.

N	K_r	K_c	T_{cm}	T_{mv}	E	M
1024	1024	1024	5.1E−1	2.0E−3	9.3E−16	8.4E+0
2048	1969	1969	3.0E+0	6.0E−3	5.6E−12	3.2E+1
4096	3285	3287	9.7E+0	1.6E−2	6.8E−11	9.8E+1
8192	5360	5362	4.4E+1	4.8E−2	6.3E−11	3.0E+2
16384	8703	8707	2.9E+2	1.5E−1	5.7E−11	9.3E+2
32768	14015	14013	1.9E+3	5.5E−1	7.5E−11	2.9E+3

respectively; at a lower precision of $\epsilon = 10^{-3}$, these become just 40 and 180 MB. Finally, to provide some intuition about the behavior of the algorithm as a function of precision, we report the following timings for the 2D volume case with $N = 131072$: for $\epsilon = 10^{-3}$, $T_{cm} = 41$ s and $T_{mv} = 0.09$ s; for $\epsilon = 10^{-6}$, $T_{cm} = 161$ s and $T_{mv} = 0.18$ s; and for $\epsilon = 10^{-9}$, $T_{cm} = 339$ s and $T_{mv} = 0.27$ s.

6.1.2. The Helmholtz equation. We next considered the 2D and 3D Helmholtz kernels

$$G(x,y) = \frac{\imath}{4} H_0^{(1)}(k|x-y|) \tag{6.1}$$

and

$$G(x,y) = \frac{e^{\imath k|x-y|}}{4\pi|x-y|}, \tag{6.2}$$

respectively, where $H_0^{(1)}$ is the zeroth order Hankel function of the first kind and k is the wavenumber. We used the same representative geometries as for the Laplace equation. The size of each domain Ω in wavelengths was given by

$$\omega = \frac{k}{2\pi} \operatorname{diam}(\Omega).$$

Timing results against LAPACK/ATLAS and the FMM at low frequency ($\omega = 10$ in 2D and $\omega = 5$ in 3D) with $\epsilon = 10^{-9}$ are shown in Figure 6.2. In this regime, the performance is very similar to that for the Laplace equation, as both kernels are essentially nonoscillatory; detailed data are therefore omitted. However, as discussed in [35], the compression efficiency deteriorates as ω increases, due to the growing ranks of the matrix blocks. In the high-frequency regime, there is no asymptotic gain in efficiency. Still, numerical results suggest that the algorithm remains viable up to $\omega \sim 200$ in 2D and $\omega \sim 10$ in 3D. In all cases, the CPU times and storage

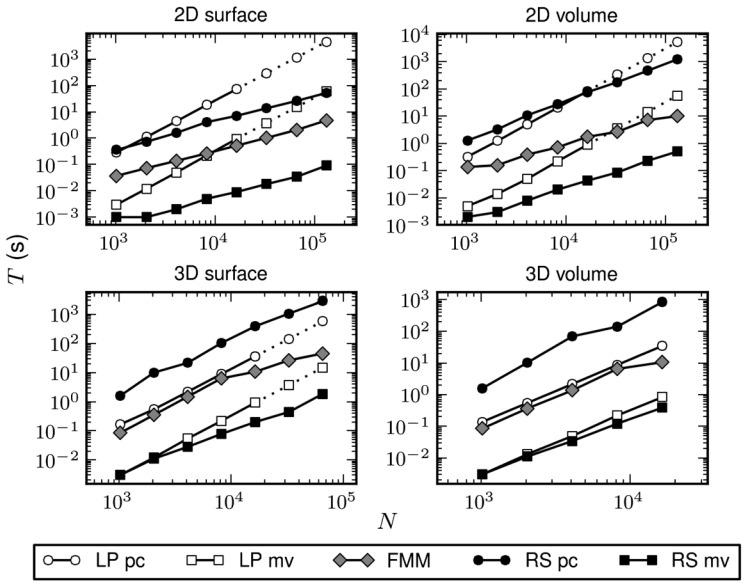

FIG. 6.2. *CPU times for applying the Helmholtz kernel in various cases at low frequency ($\omega = 10$ in 2D and $\omega = 5$ in 3D) using LAPACK/ATLAS, the FMM, and recursive skeletonization at precision $\epsilon = 10^{-9}$; notation is as in Figure* 6.1.

requirements are larger than those for the Laplace equation by a factor of about two since all computations are performed over $\mathbb{C}$ instead of $\mathbb{R}$; in 2D, there is also the additional expense of computing $H_0^{(1)}$.

6.2. Recursive skeletonization as a direct solver. In this section, we study the behavior of our algorithm as a fast direct solver. More specifically, we considered the interior Dirichlet problem for the Laplace and Helmholtz equations in 2D and 3D, recast as a second-kind boundary integral equation using the double-layer representation (2.8). Contour integrals in 2D were discretized using the trapezoidal rule, while surface integrals in 3D were discretized using Gaussian quadrature on flat triangles. In each case, we took as boundary data the field generated by an exterior point source; the error was assessed by comparing the field evaluated using the numerical solution via (2.8) against the exact field due to that source at an interior location. As a benchmark, we also solved each system directly using LAPACK/ATLAS, as well as iteratively using GMRES with matrix-vector products accelerated by the FMM.

6.2.1. The Laplace equation. For the Laplace equation (2.7), the Green function G in (2.10) is given by (2.5) in 2D and (2.9) in 3D. As a model geometry, we considered an ellipse with aspect ratio $\alpha = 2$ (semimajor and -minor axes $a = 2$ and $b = 1$, respectively) in 2D and the unit sphere in 3D; these boundaries have dimensions $d = 1$ and $d = 2$, respectively. Timing results are shown in Figure 6.3, with detailed data given in Tables 6.5 and 6.6; the precision was set to $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D.

In 2D, the solver has linear complexity and is exceptionally fast, handily beating the $\mathcal{O}(N^3)$ uncompressed direct solver, but also coming very close to the $\mathcal{O}(N)$ FMM/GMRES iterative solver. At $N = 131072$, for example, the total solution time for the recursive skeletonization algorithm was $T_{\mathrm{RS}} = 8.5$ s, while that for FMM/GMRES was $T_{\mathrm{FMM}} = 6.9$ s using $n_{\mathrm{FMM}} = 7$ iterations. It is worth emphasizing,

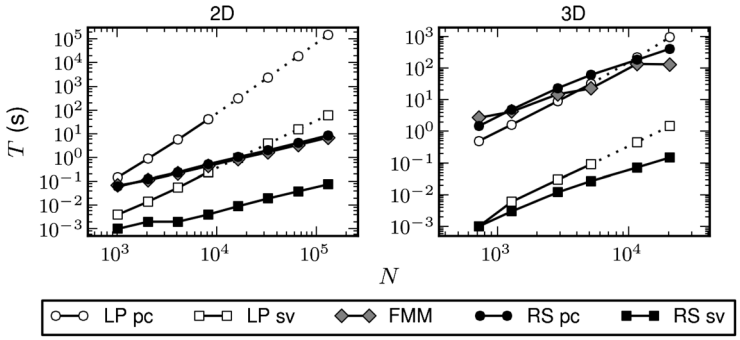

FIG. 6.3. *CPU times for solving Laplace's equation in various cases using LAPACK/ATLAS (LP), FMM/GMRES (FMM), and recursive skeletonization (RS) as a function of the system size N. For LP and RS, the computation is split into two parts: precomputation (pc), for LP consisting of matrix formation and factorization, and for RS of matrix compression and factorization; and system solution (sv), consisting of matrix inverse application. The precision of the FMM and RS was set at $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D. Dotted lines indicate extrapolated data.*

TABLE 6.5

Numerical results for solving Laplace's equation in 2D at precision $\epsilon = 10^{-9}$: N, uncompressed matrix dimension; K_{r}, row skeleton dimension; K_{c}, column skeleton dimension; T_{cm}, matrix compression time (s); T_{lu}, sparse matrix factorization time (s); T_{sv}, inverse application time (s); E, relative error; M, required storage for compressed matrix inverse (MB).

N	K_{r}	K_{c}	T_{cm}	T_{lu}	T_{sv}	E	M
1024	30	30	3.4E−2	2.5E−2	1.0E−3	9.0E−11	1.6E+0
2048	29	30	7.0E−2	5.1E−2	2.0E−3	9.0E−12	3.3E+0
4096	30	30	1.4E−1	9.8E−2	2.0E−3	8.3E−11	6.8E+0
8192	30	31	3.0E−1	2.1E−1	4.0E−3	1.6E−10	1.4E+1
16384	31	31	5.5E−1	4.5E−1	9.0E−3	5.5E−10	2.8E+1
32768	30	30	1.1E+0	8.5E−1	1.9E−2	4.9E−12	5.6E+1
65536	30	30	2.3E+0	1.8E+0	3.8E−2	1.1E−11	1.1E+2
131072	29	29	4.6E+0	3.7E+0	7.5E−2	8.5E−11	2.2E+2

however, that our solver is direct and possesses obvious advantages over FMM/GMRES, as described in section 1; in particular, the algorithm is relatively insensitive to geometric ill-conditioning. Indeed, the direct solver edged out FMM/GMRES even at modest aspect ratios (for $N = 8192$ at $\epsilon = 10^{-12}$ with $\alpha = 8$: $T_{\mathrm{RS}} = 0.76$ s, $T_{\mathrm{FMM}} = 0.98$ s, $n_{\mathrm{FMM}} = 15$); for larger α, the effect was even more pronounced ($\alpha = 512$: $T_{\mathrm{RS}} = 2.5$ s, $T_{\mathrm{FMM}} = 3.9$ s, $n_{\mathrm{FMM}} = 44$). Furthermore, the compressed inverse representation allows subsequent solves to be performed extremely rapidly; for instance, at $N = 131072$, the solve time was just $T_{\mathrm{sv}} = 0.07$ s, i.e., $T_{\mathrm{FMM}}/T_{\mathrm{sv}} \sim 100$. Thus, our algorithm is especially efficient in regimes where T_{sv} dominates (see, e.g., [34]). Finally, we remark that although direct methods are traditionally very memory-intensive, our algorithm appears quite manageable in this regard: at $N = 131072$, the storage required for the compressed inverse was only 106 MB for $\epsilon = 10^{-3}$, 172 MB for $\epsilon = 10^{-6}$, and 222 MB for $\epsilon = 10^{-9}$.

In 3D, our solver has complexity $\mathcal{O}(N^{3/2})$. Hence, asymptotics dictate that it must eventually lose. However, our results demonstrate that even up to $N = 20480$, the solver remains surprisingly competitive. For example, at $N = 20480$, $T_{\mathrm{RS}} = 409$ s, while $T_{\mathrm{FMM}} = 131$ s with $n_{\mathrm{FMM}} = 3$; at $\epsilon = 10^{-9}$, the difference is almost negligible: $T_{\mathrm{RS}} = 850$ s, $T_{\mathrm{FMM}} = 839$ s, $n_{\mathrm{FMM}} = 5$. Thus, our algorithm remains a viable alternative for medium-scale problems. It is important to note that the *solve time*

TABLE 6.6

Numerical results for solving Laplace's equation in 3D at precision $\epsilon = 10^{-6}$; notation is as in Table 6.5.

N	K_r	K_c	T_{cm}	T_{lu}	T_{sv}	E	M
720	628	669	1.3E+0	1.1E−1	1.0E−3	9.8E−5	4.6E+0
1280	890	913	4.5E+0	4.0E−1	3.0E−3	5.5E−5	1.1E+1
2880	1393	1400	2.1E+1	2.0E+0	1.2E−2	2.4E−5	5.5E+1
5120	1886	1850	5.5E+1	5.4E+0	2.7E−2	1.3E−5	1.3E+2
11520	2750	2754	1.6E+2	1.7E+1	7.2E−2	6.2E−6	3.5E+2
20480	3592	3551	3.7E+2	4.1E+1	1.5E−1	3.3E−6	6.9E+2

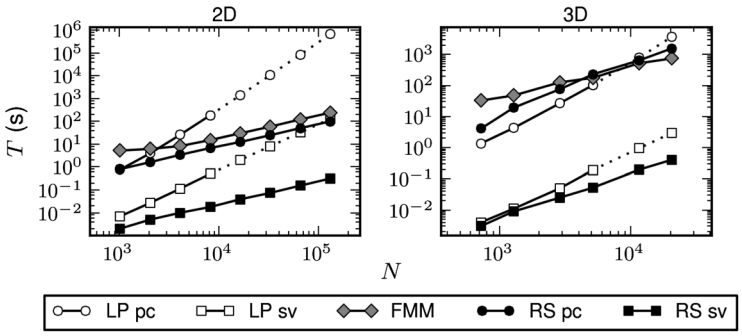

FIG. 6.4. *CPU times for solving the Helmholtz equation in various cases at low frequency ($\omega = 10$ in 2D and $\omega = 3.18$ in 3D) using LAPACK/ATLAS, FMM/GMRES, and recursive skeletonization; notation is as in Figure 6.3. The precision was set to $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D.*

advantage is not lost even for large N, since the cost of each solve is only $\mathcal{O}(N \log N)$. In fact, the advantage is, remarkably, even more striking than in 2D: at $N = 20480$, $T_{\text{FMM}}/T_{\text{sv}} \sim 1000$; for $\epsilon = 10^{-9}$, $T_{\text{FMM}}/T_{\text{sv}} \sim 2500$.

6.2.2. The Helmholtz equation. We then considered the Helmholtz equation

$$\left(\Delta + k^2\right) u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \partial\Omega,$$

recast as a boundary integral equation (2.10), with Green's function (6.1) in 2D and (6.2) in 3D. This representation does not work for all frequencies, encountering spurious discrete resonances for k beyond a critical value. We ignore that (well-understood) issue here and assume that the integral equation we obtain is invertible. The method itself does not falter in such cases, as discussed in [35].

We used the same geometries and precisions as for the Laplace equation. In 2D, the double-layer kernel is weakly singular, so we modified the trapezoidal rule with 10th-order endpoint corrections [30]. The frequency was set to $\omega = 10$ in 2D and $\omega = 3.18$ in 3D.

Timing results are shown in Figure 6.4. The data are very similar to that for the Laplace equation, but with the direct solver actually beating FMM/GMRES in 2D. This is because the number of iterations required for FMM/GMRES scales as $n_{\text{FMM}} = \mathcal{O}(\omega)$. Interestingly, even at moderately high frequencies, where we would expect the direct solver to break down as discussed in section 6.1, the performance drop is more than compensated for by the increase in the number n_{FMM} of iterations. In short, we find that recursive skeletonization is faster than FMM/GMRES at low to moderate frequencies, provided that the memory requirement is not excessive.

The story is much the same in 3D and the compressed solve time is again very fast: at $N = 20480$, $T_{\text{FMM}}/T_{\text{sv}} \sim 2000$.

6.3. Molecular electrostatics. An important application area for our solver is molecular electrostatics. A simplified model for this involves consideration of a molecular surface Σ, dividing $\mathbb{R}^3$ into Ω_1 and Ω_2, denoting the molecule and the solvent, respectively. We also suppose that the molecule has interior charges of strengths q_i at locations $x_i \in \Omega_1$ for $i = 1, \ldots, n$. The electrostatic potential φ (ignoring salt effects in the solvent) then satisfies the Poisson equation

$$-\nabla \cdot [\varepsilon(x) \nabla \varphi(x)] = \sum_{i=1}^{n} q_i \delta(x - x_i),$$

where

$$\varepsilon(x) = \begin{cases} \varepsilon_1 & \text{if } x \in \Omega_1, \\ \varepsilon_2 & \text{if } x \in \Omega_2 \end{cases}$$

is a piecewise constant dielectric. We decompose the solution as $\varphi = \varphi_s + \varphi_p$, where φ_s is the potential due to the sources,

$$\varphi_s(x) = \frac{1}{\varepsilon_1} \sum_{i=1}^{n} q_i G(x, x_i), \tag{6.3}$$

with G given by (2.9), and φ_p is a piecewise harmonic potential, satisfying the jump conditions

$$[\varphi_p] = 0, \qquad \left[\varepsilon \frac{\partial \varphi_p}{\partial \nu}\right] = -\left[\varepsilon \frac{\partial \varphi_s}{\partial \nu}\right]$$

on Σ, where ν is the unit outer normal. We can write φ_p, called the *polarization response*, as a single-layer potential [24]

$$\varphi_p(x) = \int_{\Sigma} G(x, y) \sigma(y) \, dy, \tag{6.4}$$

which yields the boundary integral equation

$$\frac{1}{2}\sigma(x) + \lambda \int_{\Sigma} \frac{\partial G}{\partial \nu_x}(x, y) \sigma(y) \, dy = -\lambda \frac{\partial \varphi_s}{\partial \nu}(x),$$

where $\lambda = (\varepsilon_1 - \varepsilon_2)/(\varepsilon_1 + \varepsilon_2)$, in terms of the *polarization charge* σ. Once σ has been computed, the potential at any point can be evaluated using (6.3) and (6.4).

We generated a molecular surface for a short segment of DNA [15, PDB ID: 1BNA] consisting of $N = 19752$ triangles using MSMS [42]. Strengths were assigned to each of $n = 486$ heavy atoms using Amber partial charges [5] through PDB2PQR [14]. The system was solved with $\varepsilon_1 = 20$ and $\varepsilon_2 = 80$ at precision $\epsilon = 10^{-3}$; the resulting potential φ on Σ is shown in Figure 6.5. The net solution time was $T_{\text{RS}} = 592$ s, with an inverse application time of $T_{\text{sv}} = 0.08$ s, to be compared with $T_{\text{FMM}} = 27$ s using FMM/GMRES. Thus, when sampling the electrostatic field for many charge configurations $\{q_i\}$, as is common in computational chemistry (e.g., [3]), our solver can provide a speed-up as long as the number of such configurations is greater than $\sim$25. We remark that the evaluation of φ at fixed points, e.g., on Σ, via (6.3) and (6.4) can also be accelerated using our algorithm in its capacity as a generalized FMM; the computation time for this would be similar to T_{sv}.

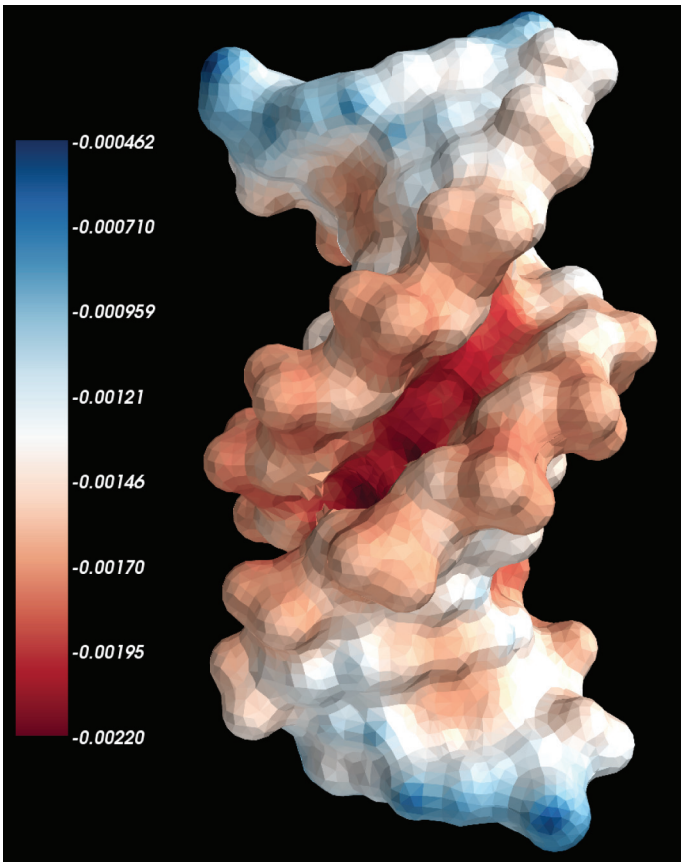

FIG. 6.5. *Surface potential of DNA (PDB ID: 1BNA) in units of the elementary charge, computed using recursive skeletonization to precision* $\epsilon = 10^{-3}$. *The molecular surface was discretized using* $N = 19752$ *triangles.*

6.4. Multiple scattering. As a final example, we show how direct solvers can be combined with FMM-based iterative methods to great effect in the context of a multiple scattering problem. For this, let Ω_i, for $i = 1, \ldots, p$, be a collection of acoustic scatterers in 2D with boundaries Σ_i. Then, using the language of acoustics, the pressure field satisfies

$$\left(\Delta + k^2\right) u = 0 \quad \text{in } \mathbb{R}^2 \setminus \bigcup_{i=1}^{p} \Omega_i. \tag{6.5}$$

Assuming that the obstacles are *sound-hard*, we must compute the exterior solution that satisfies the Neumann boundary condition

$$\frac{\partial u}{\partial \nu} = 0 \quad \text{on } \bigcup_{i=1}^{p} \Sigma_i.$$

If $u = u_i + u_s$, where u_i is an incoming field satisfying (6.5), then the scattered field u_s also satisfies (6.5) with boundary condition

$$\frac{\partial u_s}{\partial \nu} = -\frac{\partial u_i}{\partial \nu} \quad \text{on } \bigcup_{i=1}^{p} \Sigma_i.$$

We write the scattered field as $u_s = \sum_{i=1}^{p} u_{s,i}$, where

$$u_{s,i}(x) = \int_{\Sigma_i} G(x,y)\,\sigma_i(y)\,dy,$$

where G is the single-layer kernel (6.1). Imposing the boundary condition yields the second-kind integral equation

$$-\frac{1}{2}\sigma_i + \sum_{j=1}^{p} K_{ij}\sigma_j = -\left.\frac{\partial u_i}{\partial \nu}\right|_{\Sigma_i} \quad \text{on } \Sigma_i, \quad i = 1,\ldots,p,$$

where

$$K_{ij}\sigma_j(x) = \int_{\Sigma_j} \frac{\partial G}{\partial \nu_x}(x,y)\,\sigma_j(y)\,dy \quad \text{for } x \in \Sigma_i.$$

In operator notation, the linear system therefore has the form

$$\sum_{i=1}^{p} A_{ij}\sigma_j = -\left.\frac{\partial u_i}{\partial \nu}\right|_{\Sigma_i}, \quad A_{ij} = \begin{cases} -\frac{1}{2}I + K_{ii} & \text{if } i = j, \\ K_{ij} & \text{if } i \neq j. \end{cases}$$

We solve this system using FMM/GMRES with the block diagonal preconditioner

$$P^{-1} = \begin{bmatrix} A_{11}^{-1} & & \\ & \ddots & \\ & & A_{pp}^{-1} \end{bmatrix},$$

where each A_{ii}^{-1} is computed using recursive skeletonization; observe that A_{ii}^{-1} is precisely the solution operator for scatterer Ω_i in isolation. The question is whether this preconditioner will significantly reduce the iteration count required, which is typically quite high for problems of appreciable size.

As a test, we embedded two identical scatterers, each described in polar coordinates by the radial function $r = [2 + \cos(3\theta)]/6$, where θ is the polar angle; each scatterer is smooth, though somewhat complicated, and was taken to be 10 wavelengths in size. We assumed an incoming field given by the plane wave $u_i = e^{\imath k x_2}$, where $x = (x_1, x_2)$, and considered the scattering problem at various horizontal separation distances δ between the centers of the scatterers. Each configuration was solved both with and without the preconditioner P^{-1} to precision $\epsilon = 10^{-6}$; each scatterer was discretized using a corrected trapezoidal rule [30] with $N = 1024$ points.

The intensities of the resulting pressure fields are shown in Figure 6.6, with numerical data given in Table 6.7. It is clear that the preconditioner is highly effective: following a precomputation time of 0.76 s to construct P^{-1}, which is amortized over all solves, the number of iterations required was decreased from $n_{\mathrm{FMM}} \sim 700$ to just $n_{\mathrm{RS}} \sim 10$ for each case. As expected, more iterations were necessary for smaller δ, though the difference was not too dramatic. The ratio of the total solution time required for all solves was ~ 60 for the unpreconditioned versus the preconditioned method.

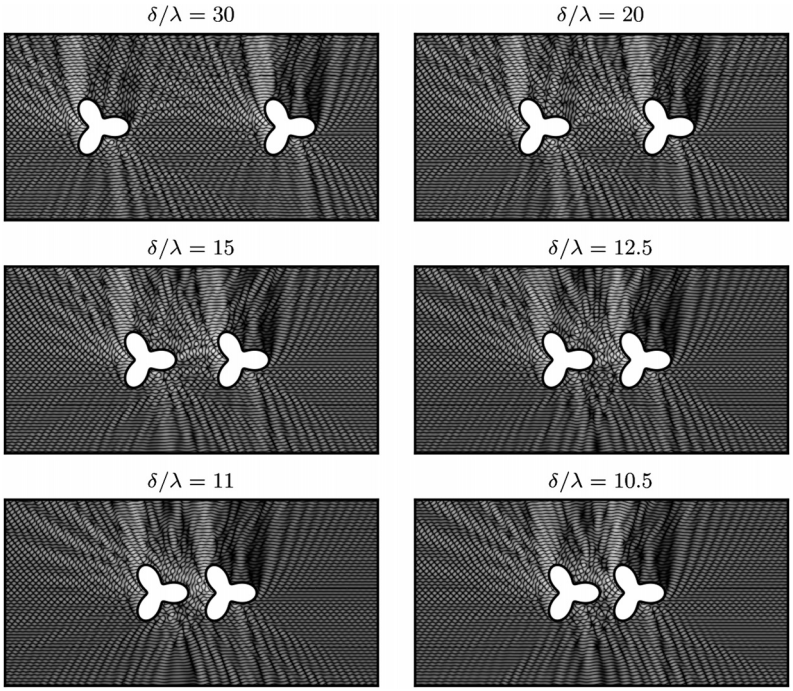

FIG. 6.6. *Instantaneous intensity* $[\Re(u)]^2$ *of the pressure field in response to an incoming vertical plane wave for various scattering geometries characterized by the separation distance* δ/λ *in wavelengths between the centers of two identical scatterers.*

TABLE 6.7

Numerical results for the multiple scattering example, consisting of six configurations with various separation distances δ/λ*, relative to the wavelength, between the centers of two identical scatterers, solved to precision* $\epsilon = 10^{-6}$*:* T_{FMM}*, time for FMM/GMRES solve (s);* T_{RS}*, time for preconditioned FMM/GMRES solve (s);* n_{FMM}*, number of iterations required for FMM/GMRES;* n_{RS}*, number of iterations required for preconditioned FMM/GMRES;* E*, relative error;* T_{cm}*, matrix compression time for scatterer (s);* T_{lu}*, sparse matrix factorization time for scatterer (s).*

δ/λ	T_{FMM}	T_{RS}	n_{FMM}	n_{RS}	E
30.0	7.9E+1	8.9E−1	697	8	1.3E−8
20.0	7.7E+1	1.1E+0	694	10	5.8E−9
15.0	8.0E+1	1.2E+0	695	11	6.9E−9
12.5	7.9E+1	1.3E+0	695	12	7.8E−9
11.0	7.9E+1	1.4E+0	704	14	8.7E−9
10.5	8.0E+1	1.5E+0	706	14	1.3E−8
T_{cm}		6.6E−1			
T_{lu}		9.3E−2			
total	4.7E+2	8.1E+0			

7. Generalizations and conclusions. We have presented a multilevel matrix compression algorithm and demonstrated its efficiency at accelerating matrix-vector multiplication and matrix inversion in a variety of contexts. The matrix structure required is fairly general and relies only on the assumption that the matrix have low-rank off-diagonal blocks. As a fast direct solver for the boundary integral equations of potential theory, we found our algorithm to be competitive with fast iterative methods based on FMM/GMRES in both 2D and 3D, provided that the integral equation kernel is not too oscillatory and that the system size is not too large in 3D. In such cases, the total solution times for both methods were very comparable. Our solver has clear

advantages, however, for problems with ill-conditioned matrices (in which case the number of iterations required by FMM/GMRES can increase dramatically), or those involving multiple right-hand sides (in which case the cost of matrix compression and factorization can be amortized). The latter category includes the use of our solver as a preconditioner for iterative methods, which we expect to be quite promising, particularly for large-scale 3D problems with complex geometries.

A principal limitation of the approach described here is the growth in the cost of factorization in 3D or higher, which prohibits the scheme from achieving optimal $\mathcal{O}(N)$ or nearly optimal $\mathcal{O}(N \log N)$ complexity. It is, however, straightforward to implement and quite effective. All of the hierarchical compression-based approaches (HSS matrices [6, 7, 49], $\mathcal{H}$-matrices [26, 27, 28], and skeletonization [17, 21, 35]) are capable of overcoming this obstacle. The development of simple and effective schemes that curtail this growth is an active area of research, and we expect that $\mathcal{O}(N \log N)$ direct solvers with small prefactors in higher dimensions will be constructed shortly, at least for nonoscillatory problems. It is clear that all these techniques provide improved solution times for high-frequency *volume* integral equations, due to the compression afforded by Green's theorem in moving data from the volume to the boundary. More precisely, the cost of solving high-frequency volume wave scattering problems in 2D are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for precomputation and solution, respectively. For related work, see [8, 47].

Finally, although all numerical results have presently been reported for a single processor, the algorithm is naturally parallelizable: many computations are organized in a block sweep structure, where each block can be processed independently. This is clearly true of the recursive skeletonization phase using proxy surfaces (with a possible loss of $\mathcal{O}(\log N)$ in performance since there are $\mathcal{O}(\log N)$ levels in the hierarchy). As for the solver phase, arguments can be made in support of both the original hand-rolled Gaussian elimination approach and our framework that relies on sparse embedding. We expect that by making use of UMFPACK and other state-of-the-art parallel sparse solvers (e.g., SuperLU [31], MUMPS [1], Pardiso [43], WSMP [25]), our overall strategy will help simplify the implementation of skeletonization-based schemes on high-performance computing platforms as well.

Acknowledgments. We would like to thank Zydrunas Gimbutas and Mark Tygert for many helpful discussions.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[3] P. Beroza, D. R. Fredkin, M. Y. Okamura, and G. Feher, *Protonation of interacting residues in a protein by a Monte Carlo method: Application to lysozyme and the photosynthetic reaction center of Rhodobacter sphaeroides*, Proc. Natl. Acad. Sci. USA, 88 (1991), pp. 5804–5808.

[4] J. Bremer, *A fast direct solver for the integral equations of scattering theory on planar curves with corners*, J. Comput. Phys., 231 (2012), pp. 1879–1899.

[5] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, *The Amber biomolecular simulation programs*, J. Comput. Chem., 26 (2005), pp. 1668–1688.

[6] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.

[7] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[8] Y. Chen, *A fast, direct algorithm for the Lippmann-Schwinger integral equation in two dimensions*, Adv. Comput. Math., 16 (2002), pp. 175–190.

[9] H. Cheng, W. Y. Crutchfield, Z. Gimbutas, L. F. Greengard, J. F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao, *A wideband fast multipole method for the Helmholtz equation in three dimensions*, J. Comput. Phys., 216 (2006), pp. 300–325.

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

[11] W. C. Chew, J.-M. Jin, E. Michielssen, and J. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, Boston, MA, 2001.

[12] T. A. Davis, *Algorithm* 832*: UMFPACK v*4.3*—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Softw., 30 (2004), pp. 196–199.

[13] T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 140–158.

[14] T. J. Dolinsky, J. E. Nielsen, J. A. McCammon, and N. A. Baker, *PDB2PQR: An automated pipeline for the setup of Poisson-Boltzmann electrostatics calculations*, Nucleic Acids Res., 32 (2004), pp. W665–W667.

[15] H. R. Drew, R. M. Wing, T. Takano, C. Broka, S. Tanaka, K. Itakura, and R. E. Dickerson, *Structure of a B-DNA dodecamer: conformation and dynamics*, Proc. Natl. Acad. Sci. USA, 78 (1981), pp. 2179–2183.

[16] A. Gillman, *Fast Direct Solvers for Elliptic Partial Differential Equations*, Ph.D. thesis, Department of Applied Mathematics, University of Colorado at Boulder, 2011.

[17] A. Gillman, P. Young, and P.G. Martinsson, *A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247.

[18] Z. Gimbutas and L. Greengard, *FMMLIB: fast multipole methods for electrostatics, elastostatics, and low frequency acoustic modeling*, in preparation, Software available from http://www.cims.nyu.edu/cmcl/software.html.

[19] Z. Gimbutas and V. Rokhlin, *A generalized fast multipole method for nonoscillatory kernels*, SIAM J. Sci. Comput., 24 (2003), pp. 796–817.

[20] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[21] L. Greengard, D. Gueyffier, P.-G. Martinsson, and V. Rokhlin, *Fast direct solvers for integral equations in complex three-dimensional domains*, Acta Numer., 18 (2009), pp. 243–275.

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

[23] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., 6 (1997), pp. 229–269.

[24] R. B. Guenther and J. W. Lee, *Partial Differential Equations of Mathematical Physics and Integral Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[25] A. Gupta, *WSMP: Watson Sparse Matrix Package. Part II—Direct Solution of General Sparse Systems*, Technical report RC 21888, IBM T. J. Watson Research Center, 2000.

[26] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I*: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[27] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[28] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part* II*: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[29] W. W. Hager, *Updating the inverse of a matrix*, SIAM Rev., 31 (1989), pp. 221–239.

[30] S. Kapur and V. Rokhlin, *High-order corrected trapezoidal quadrature rules for singular functions*, SIAM J. Numer. Anal., 34 (1997), pp. 1331–1356.

[31] X. S. Li, *An overview of SuperLU: algorithms, implementation, and user interface*, ACM Trans. Math. Softw., 31 (2005), pp. 302–325.

[32] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.

[33] Y. Liu, *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*, Cambridge University Press, New York, 2009.

[34] P.-G. Martinsson, *Fast evaluation of electro-static interactions in multi-phase dielectric media*, J. Comput. Phys., 211 (2006), pp. 289–299.
[35] P.-G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
[36] P.-G. Martinsson and V. Rokhlin, *An accelerated kernel-independent fast multipole method in one dimension*, SIAM J. Sci. Comput., 29 (2007), pp. 1160–1178.
[37] P.-G. Martinsson and V. Rokhlin, *A fast direct solver for scattering problems involving elongated structures*, J. Comput. Phys., 221 (2007), pp. 288–302.
[38] N. Nishimura, *Fast multipole accelerated boundary integral equation methods*, Appl. Mech. Rev., 55 (2002), pp. 299–324.
[39] T. P. Pals, *Multipole for Scattering Computations: Spectral Discretization, Stabilization, Fast Solvers*, Ph.D. thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, 2004.
[40] V. Rokhlin, *Rapid solution of integral equations of scattering theory in two dimensions*, J. Comput. Phys., 86 (1990), pp. 414–439.
[41] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
[42] M. F. Sanner, A. J. Olson, and J.-C. Spehner, *Reduced surface: An efficient way to compute molecular surfaces*, Biopolymers, 38 (1996), pp. 305–320.
[43] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Gener. Comput. Syst., 20 (2004), pp. 475–487.
[44] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[45] J.-G. Wei, Z. Peng, and J.-F. Lee, *A fast direct matrix solver for surface integral equation methods for electromagnetic wave problems in* $\mathcal{R}^3$, in Proceedings of the 27th International Review of Progress in Applied Computational Electromagnetics, Williamsburg, VA, 2011, pp. 121–126.
[46] R. C. Whaley, A. Petitet, and J. J. Dongarra, *Automated empirical optimization of software and the ATLAS project*, Parallel Comput., 27 (2001), pp. 3–35.
[47] E. Winebrand and A. Boag, *A multilevel fast direct solver for EM scattering from quasi-planar objects*, in Proceedings of the International Conference on Electromagnetics in Advanced Applications, Torino, Italy, 2009, pp. 640–643.
[48] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.
[49] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.
[50] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626.