INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2003; **56**:239–260 (DOI: 10.1002/nme.562)

A fast boundary cloud method for exterior 2D electrostatic analysis

Vaishali Shrivastava and N. R. Aluru*,†

Department of General Engineering, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, U.S.A.

SUMMARY

An accelerated boundary cloud method (BCM) for boundary-only analysis of exterior electrostatic problems is presented in this paper. The BCM uses scattered points instead of the classical boundary elements to discretize the surface of the conductors. The dense linear system of equations generated by the BCM are solved by a GMRES iterative solver combined with a singular value decomposition based rapid matrix–vector multiplication technique. The accelerated technique takes advantage of the fact that the integral equation kernel (2D Green's function in our case) is locally smooth and, therefore, can be dramatically compressed by using a singular value decomposition technique. The acceleration technique greatly speeds up the solution phase of the linear system by accelerating the computation of the dense matrix–vector product and reducing the storage required by the BCM. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: meshless methods; moving least squares; boundary cloud method (BCM); fast algorithms; singular value decomposition (SVD)

1. INTRODUCTION

Computational analysis of micro-electro-mechanical systems (MEMS) requires an accurate analysis of exterior electrostatic problems [1–5]. Boundary element methods [6] are best suited for exterior electrostatic analysis as they require meshing of only the surface of the conductor. Even though the meshing of surfaces can be less involved compared to the meshing of entire volumes, it can still be quite involved. An attractive alternative is to combine meshless techniques (see References [7–16] and other references in these papers) with boundary integral formulations to alleviate meshing requirements. Boundary-type methods [17, 18] that combine boundary integral formulations with meshless techniques have been developed. We have recently introduced a boundary cloud method (BCM) [19] that also combines boundary

*Correspondence to: N. R. Aluru, Department of General Engineering, Beckman Institute for Advanced Science and Technology, University of Illionis at Urbana-Champaign, Urbana, IL 61801, U.S.A.
†E-mail: aluru@uiuc.edu

Contract/grant sponsor: NSF CAREER.

Received 31 August 2001
Revised 13 February 2002
Accepted 20 March 2002

integral formulations with meshless techniques. The key advantage with a BCM method is that one can use Cartesian co-ordinates to compute meshless interpolation functions.

Boundary element methods, as well as the meshless boundary integral formulations such as the BNM and BCM, generate dense matrices to compute unknown quantities. The solution of dense linear systems can be quite involved. First, they require $O(N^2)$ storage, where N is the number of nodes or panels. Second, the use of classical direct methods, such as the Gaussian elimination, requires $O(N^3)$ operations. Iterative methods can be less expensive, but they still require the calculation of a dense matrix–vector product, which costs about $O(N^2)$ operations. Typically, an iterative technique requires the calculation of several dense matrix–vector products before a converged solution is obtained. Several approaches have been proposed to accelerate the calculation of dense matrix–vector products in iterative solvers. The first approach is the fast multipole method (FMM) [20], which was originally developed for particle simulation problems. Using an FMM, the dense matrix–vector product can be computed in $O(mN)$ time and memory, where m is the number of conductors and N is the number of panels used to discretize a conductor. The central strategy used in FMM is that of clustering particles at various spatial lengths and computing interactions with other clusters which are sufficiently far away by means of multipole expansions. The original implementation of FMM was tailored to the $1/\|x - x'\|$ kernel. The second approach is the precorrected fast Fourier transform (FFT) technique [21]. The central idea in the precorrected FFT technique is to represent the long-range part of the potential by point charges lying on a uniform grid, rather than by a series expansion as was done in the fast multipole method. It is based on the availability of efficient discrete Fourier transform. The complexity of the precorrected FFT algorithm is $O(N \log N)$. Recently, a third method, which uses the concept of singular value decomposition (SVD), is introduced in References [22, 23] to compute the dense matrix–vector products. This approach utilizes the fact that large parts of integral operator matrix are numerically low rank. SVD can be used to compress these rank-deficient matrices. The complexity of this approach is also $O(N \log N)$.

In this paper, we introduce a fast boundary method, which combines the SVD-based fast algorithm with the BCM. Compared to the other fast algorithms, SVD-based acceleration technique is much easier to implement as it operates directly on the dense matrix. Even though [22, 23] mention that the complexity of the algorithm is $O(N \log N)$ for three-dimensional (3D) problems, our results indicate that the complexity of the approach can be $O(N(\log N)^2)$ for two-dimensional (2D) problems, especially when the dense matrix involves predominantly near-by interactions (see Appendix A). The rest of the paper is outlined as follows: Section 2 describes the BCM, Section 3 explains the SVD-based acceleration technique for solving the dense linear system, Section 4 presents results for three 2D examples and conclusions are presented in Section 5.

2. BOUNDARY CLOUD METHOD FOR EXTERIOR ELECTROSTATICS

2.1. Governing equation and boundary integral formulation

Consider a 2D two-conductor system as shown in Figure 1. Ω_1 and Ω_2 are the domains occupied by the two conductors, Γ_1 and Γ_2 are the boundaries of Ω_1 and Ω_2, respectively, d is the separation between the two conductors and $\bar{\Omega}$ is the domain exterior to the two bodies

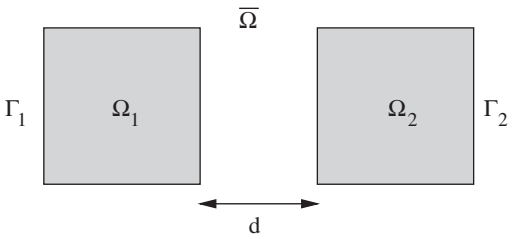

Figure 1. A two conductor electrostatic system.

or conductors. The governing equation for the exterior electrostatic problem is [6]

$$\nabla^2\phi = 0 \quad \text{in } \bar{\Omega} \tag{1}$$

$$\phi_1 = g_1 \quad \text{on } \Gamma_1 \tag{2}$$

$$\phi_2 = g_2 \quad \text{on } \Gamma_2 \tag{3}$$

where ϕ is the electrostatic potential of the conductors and g_1 and g_2 are the prescribed electrostatic potentials.

The boundary integral equation for the electrostatic problem is given by [6]

$$\phi(p) = \int_{\mathrm{d}\Omega} \frac{1}{\varepsilon} G(p,q)\sigma(q)\,\mathrm{d}\gamma_q + C \tag{4}$$

$$\int_{\mathrm{d}\Omega} \sigma(q)\,\mathrm{d}\gamma_q = C_T \tag{5}$$

where $\mathrm{d}\Omega$ is the boundary of the conductors, ε is the dielectric constant of the medium, p is the source point, q is the field point which moves along the boundary of the conductors and G is the Green's function. In two dimensions $G = \ln|p-q|/2\pi$, where $|p-q|$ is the distance between the source point p and the field point q and C is an unknown variable. The unknown charge density σ is computed by employing a BCM [24]. The key steps in the boundary cloud method are described below.

2.2. *Construction of interpolation functions*

In a BCM, the surface of the domain is discretized into scattered points. The points can be sprinkled randomly covering the boundary of the domain. Interpolation functions are constructed by centring a weighting function at each point or node. The unknown quantity u and its normal derivative q are approximated by a Hermite-type interpolation. For a 2D problem, as shown in Figure 2, given a point t, the unknown and its normal derivative in the vicinity of point t are approximated by

$$u(x,y) = \mathbf{p}^{\mathrm{T}}(x,y)\mathbf{a}_t \tag{6}$$

$$q(x,y) = \frac{\partial \mathbf{p}^{\mathrm{T}}(x,y)}{\partial n}\,\mathbf{a}_t \tag{7}$$

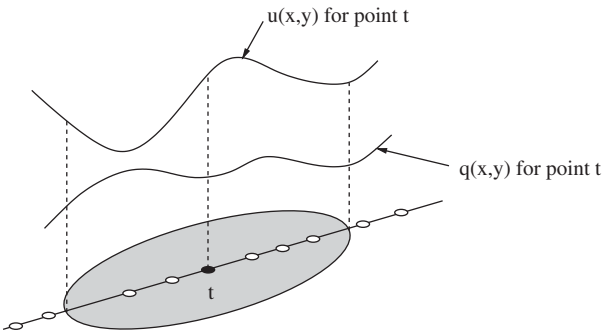

Figure 2. Interpolation region for point t.

where $\mathbf{p}$ is the base interpolating polynomial, $\mathbf{a}_t$ is the unknown coefficient vector for point t and n is the direction of the outward normal to the boundary (note that n can be different at every point). $\mathbf{a}_t$ has been assumed constant for a given point t. However, $\mathbf{a}_t$ changes for different points. In this paper, we use a linear polynomial basis. The base interpolating polynomial and their normal derivatives are given by

$$\mathbf{p}^{\mathrm{T}}(x, y) = [1 \;\; x \;\; y], \quad m = 3 \tag{8}$$

$$\frac{\partial \mathbf{p}^{\mathrm{T}}(x, y)}{\partial n} = \left[0 \;\; \frac{\partial x}{\partial n} \;\; \frac{\partial y}{\partial n}\right] = [0 \;\; \cos(\widehat{nx}) \quad \cos(\widehat{ny})], \quad m = 3 \tag{9}$$

where $\widehat{nx}$ and $\widehat{ny}$ are the angles between the outward normal direction and the positive x- and y-axis, respectively.

For a point t, the unknown coefficient vector $\mathbf{a}_t$ is computed by minimizing the following form:

$$\mathbf{J}_t = \sum_{i=1}^{\mathrm{NP}} w_i(x_t, y_t)[\mathbf{p}^{\mathrm{T}}(x_i, y_i)\mathbf{a}_t - \hat{u}_i]^2 + \sum_{i=1}^{\mathrm{NP}} w_i(x_t, y_t)\left[\frac{\partial \mathbf{p}^{\mathrm{T}}(x_i, y_i)}{\partial n}\mathbf{a}_t - \hat{q}_i\right]^2 \tag{10}$$

where NP is the number of nodes, $w_i(x_t, y_t)$ is the weighting function centred at (x_t, y_t) and evaluated at node i whose co-ordinates are (x_i, y_i). $\hat{u}_i$ and $\hat{q}_i$ are nodal parameters. The weighting function is non-zero when the location of node i is within a certain distance from the point (x_t, y_t). The region where the weighting function is non-zero is called a cloud. The weighting function is typically a cubic spline or a Gaussian function. Extensive details on the development and application of a BCM for potential problems can be found in Reference [19].

The stationary of J_t leads to

$$(\mathbf{P}^{\mathrm{T}}\bar{\mathbf{W}}\mathbf{P} + \mathbf{P}'^{\mathrm{T}}\bar{\mathbf{W}}\mathbf{P}')\mathbf{a}_t = \mathbf{P}^{\mathrm{T}}\bar{\mathbf{W}}\hat{\mathbf{u}} + \mathbf{P}'\bar{\mathbf{W}}\hat{\mathbf{q}} \tag{11}$$

Equation (11) can be rewritten as

$$\bar{\mathbf{C}}_t\bar{\mathbf{a}}_t = \bar{\mathbf{A}}_t\hat{\mathbf{u}} + \bar{\mathbf{B}}_t\hat{\mathbf{q}} \tag{12}$$

$$\mathbf{a}_t = \bar{\mathbf{C}}_t^{-1}\bar{\mathbf{A}}_t\hat{\mathbf{u}} + \bar{\mathbf{C}}_t^{-1}\bar{\mathbf{B}}_t\hat{\mathbf{q}} \tag{13}$$

where $\bar{\mathbf{C}}_t$ is an $m \times m$ matrix, $\bar{\mathbf{A}}_t$ is an $m \times \mathrm{NP}$ matrix and $\bar{\mathbf{B}}_t$ is an $m \times \mathrm{NP}$ matrix defined as

$$\bar{\mathbf{C}}_t = \mathbf{P}^{\mathrm{T}}\bar{\mathbf{W}}\mathbf{P} + \mathbf{P}'^{\mathrm{T}}\bar{\mathbf{W}}\mathbf{P}' \tag{14}$$

$$\bar{\mathbf{A}}_t = \mathbf{P}^{\mathrm{T}}\bar{\mathbf{W}} \tag{15}$$

$$\bar{\mathbf{B}}_t = \mathbf{P}'^{\mathrm{T}}\bar{\mathbf{W}} \tag{16}$$

$\mathbf{P}$ and $\mathbf{P}'$ are $\mathrm{NP} \times m$ matrices, $\hat{\mathbf{u}}$ and $\hat{\mathbf{q}}$ are $\mathrm{NP} \times 1$ vectors, $\bar{\mathbf{W}}$ is a square $\mathrm{NP} \times \mathrm{NP}$ diagonal matrix and their definitions are given by

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}^{\mathrm{T}}(x_1, y_1) \\ \mathbf{p}^{\mathrm{T}}(x_2, y_2) \\ \vdots \\ \mathbf{p}^{\mathrm{T}}(x_{\mathrm{NP}}, y_{\mathrm{NP}}) \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} \dfrac{\partial \mathbf{p}^{\mathrm{T}}(x_1, y_1)}{\partial n_1} \\ \dfrac{\partial \mathbf{p}^{\mathrm{T}}(x_2, y_2)}{\partial n_2} \\ \vdots \\ \dfrac{\partial \mathbf{p}^{\mathrm{T}}(x_{\mathrm{NP}}, y_{\mathrm{NP}})}{\partial n_{\mathrm{NP}}} \end{bmatrix} \tag{17}$$

$$\hat{\mathbf{u}} = \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_{\mathrm{NP}} \end{bmatrix}, \quad \hat{\mathbf{q}} = \begin{bmatrix} \hat{q}_1 \\ \hat{q}_2 \\ \vdots \\ \hat{q}_{\mathrm{NP}} \end{bmatrix} \tag{18}$$

$$\bar{\mathbf{W}} = \begin{bmatrix} w_1(x_t, y_t) & 0 & 0 & 0 \\ 0 & w_2(x_t, y_t) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_{\mathrm{NP}}(x_t, y_t) \end{bmatrix} \tag{19}$$

In Equation (17), $n_1, n_2, \ldots, n_{\mathrm{NP}}$ are the outward normal directions of nodes $1, 2, \ldots, \mathrm{NP}$, respectively. Substituting the definition of $\mathbf{a}_t$ into Equations (6) and (7), the unknown u and its normal derivative q, can be expressed as

$$u(x, y) = \bar{\mathbf{M}}(x, y)\hat{\mathbf{u}} + \bar{\mathbf{N}}(x, y)\hat{\mathbf{q}} \tag{20}$$

$$q(x, y) = \bar{\mathbf{S}}(x, y)\hat{\mathbf{u}} + \bar{\mathbf{T}}(x, y)\hat{\mathbf{q}} \tag{21}$$

where $\bar{\mathbf{M}}(x, y)$, $\bar{\mathbf{N}}(x, y)$, $\bar{\mathbf{S}}(x, y)$ and $\bar{\mathbf{T}}(x, y)$ are $1 \times \mathrm{NP}$ vectors defined as

$$\bar{\mathbf{M}}(x, y) = \mathbf{p}^{\mathrm{T}}(x, y)\bar{\mathbf{C}}_t^{-1}\bar{\mathbf{A}}_t \tag{22}$$

$$\bar{\mathbf{N}}(x, y) = \bar{\mathbf{C}}_t^{-1}\bar{\mathbf{B}}_t \tag{23}$$

$$\bar{\mathbf{S}}(x, y) = \frac{\partial \mathbf{p}(x, y)}{\partial n} \bar{\mathbf{C}}_t^{-1} \bar{\mathbf{A}}_t \tag{24}$$

$$\bar{\mathbf{T}}(x, y) = \frac{\partial \mathbf{p}^{\mathrm{T}}(x, y)}{\partial n} \bar{\mathbf{C}}_t^{-1} \bar{\mathbf{B}}_t \tag{25}$$

In a truncated cloud approach, where a weighting function centred at a node of an edge does not include nodes from other edges (see Reference [19] for details), we can show that $\bar{\mathbf{N}}(x, y) = 0$ and $\bar{\mathbf{S}}(x, y) = 0$. Therefore, Equations (20) and (21) can be rewritten as

$$\mathbf{u} = \bar{\mathbf{M}}\hat{\mathbf{u}} \tag{26}$$

$$\mathbf{q} = \bar{\mathbf{T}}\hat{\mathbf{q}} \tag{27}$$

At any point (x, y)

$$u(x, y) = \sum_{I=1}^{\mathrm{NP}} \bar{M}_I(x, y)\hat{u}_I \tag{28}$$

$$q(x, y) = \sum_{I=1}^{\mathrm{NP}} \bar{T}_I(x, y)\hat{q}_I \tag{29}$$

where

$$\bar{M}_I(x, y) = \mathbf{p}^{\mathrm{T}}(x, y)\bar{\mathbf{C}}_t^{-1}\mathbf{p}(x_I, y_I)w_I(x_t, y_t) \tag{30}$$

$$\bar{T}_I(x, y) = \frac{\partial \mathbf{p}^{\mathrm{T}}(x, y)}{\partial n} \bar{\mathbf{C}}_t^{-1}\mathbf{p}'(x_I, y_I)w_I(x_t, y_t) \tag{31}$$

2.3. Discretization

Let the boundary be discretized into NC cells. The boundary integral equation for the electrostatic problem given in Equations (4) and (5) can be rewritten as

$$\phi(P) = \sum_{k=1}^{\mathrm{NC}} \int_{\mathrm{d}\Omega_k} \frac{1}{2\pi\varepsilon} \ln|P - Q_k|\sigma(Q_k)\,\mathrm{d}\Gamma_{Q_k} + C \tag{32}$$

$$\sum_{k=1}^{\mathrm{NC}} \int_{\mathrm{d}\Omega_k} \sigma(Q_k)\,\mathrm{d}\Gamma_{Q_k} = C_T \tag{33}$$

where NC is the number of cells, $\mathrm{d}\Omega_k$ is the length of the kth cell, Q_k is the field point on the kth cell and $\sigma(Q_k)$ is the unknown charge density approximated by the interpolation given in Equation (29). The integrand in Equation (32) is log singular. Details on the evaluation of the integrals in Equations (32) and (33) can be found in Reference [19].

Equations (32) and (33) can be rewritten in a matrix form as

$$\begin{bmatrix} \mathbf{M}_{\mathrm{NP}\times\mathrm{NP}} & \mathbf{1}_{\mathrm{NP}\times 1} \\ \mathbf{K}_{1\times\mathrm{NP}} & 0_{1\times 1} \end{bmatrix} \begin{Bmatrix} \hat{\sigma}_{\mathrm{NP}\times 1} \\ C_{1\times 1} \end{Bmatrix} = \begin{Bmatrix} \phi_{\mathrm{NP}\times 1} \\ C_{T1\times 1} \end{Bmatrix} \tag{34}$$

where

$$\mathbf{M}_{ij} = \sum_{k=1}^{\mathrm{NC}} \int_{\mathrm{d}\Omega_k} \frac{1}{2\pi\varepsilon} \ln|P_i - Q_k| \sum_{j=1}^{\mathrm{NP}} \bar{T}_j(Q_k)\, \mathrm{d}\Gamma_{Q_k} \tag{35}$$

and

$$\mathbf{K}_{1j} = \sum_{k=1}^{\mathrm{NC}} \int_{\mathrm{d}\Omega_k} \sum_{j=1}^{\mathrm{NP}} \bar{T}_j(Q_k)\, \mathrm{d}\Gamma_{Q_k} \tag{36}$$

ϕ and $\hat{\sigma}$ are NP $\times$ 1 right-hand side and unknown vectors, respectively.

The next step is to solve the dense linear system given in Equation (34). Since $\mathbf{M}$ is a dense matrix the storage is $O(\mathrm{NP}^2)$. Direct solution of Equation (34) via Gauss elimination requires $O(\mathrm{NP}^3)$ time and hence, it is impractical for large problems. In this paper, therefore, we use GMRES [25], an iterative solver. The computationally intensive step in an iterative solver is to compute the matrix–vector product. For dense matrices this costs about $O(\mathrm{NP}^2)$ operations. Depending on the number of iterations required by the GMRES to converge, a number of dense matrix–vector products need to be computed before a solution to Equation (34) is obtained.

Using the acceleration technique described in the next section the dense matrix–vector product can be computed in at most $O(\mathrm{NP}(\log \mathrm{NP})^2)$ operations. In addition, the storage can also be reduced to $O(\mathrm{NP}(\log \mathrm{NP})^2)$ instead of the classical $O(\mathrm{NP}^2)$ requirement.

3. ACCELERATION TECHNIQUE

3.1. Basic idea

The idea behind the acceleration technique [22, 23] is to exploit the fact that the assembled matrix $\mathbf{M}$ has large sections which are rank deficient. The reason behind this is that, typical Green's functions vary smoothly. Each entry in the assembled matrix denotes interaction between two points—a source point and a field point. If the distance between them is relatively large, the influence of a source point on a far-away field point is almost the same as that of its neighbours (owing to smoothly varying Green's function). Thus, the numerical rank of such portions of the matrix is small.

SVD [24, 26, 27] can be employed to exploit the rank-deficient nature of the coefficient matrices generated by the boundary cloud method. The SVD of an $n \times n$ matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}} \tag{37}$$

where

$$\mathbf{U}\mathbf{U}^{\mathrm{T}} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I} \tag{38}$$

and

$$\mathbf{S} = \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & s_n \end{bmatrix} \tag{39}$$

with $s_1 \geqslant s_2 \geqslant \cdots \geqslant s_n \geqslant 0$. The numerical rank of a matrix to precision ε is given by integer r such that $s_r/s_1 < \varepsilon$. The matrix $\mathbf{A}$ can be approximated by

$$\tilde{\mathbf{A}} = \mathbf{U}(:, 1:r)\mathbf{S}(1:r, 1:r)\mathbf{V}^{\mathrm{T}}(1:r, :) \tag{40}$$

with

$$\|\mathbf{A} - \tilde{\mathbf{A}}\| < \varepsilon \tag{41}$$

The multiplication of a vector with matrix $\mathbf{A}$ is done by multiplying the vector in turn by $\mathbf{V}^{\mathrm{T}}(1:r, :)$, $\mathbf{S}(1:r, 1:r)$ and $\mathbf{U}(:, 1:r)$. The number of operations is $O((2n+1)r)$ and for $r \ll n$ this process is more efficient compared to the $O(n^2)$ operations that are normally required to compute the dense matrix–vector product.

The concept of SVD has been used to explain the theory behind this method. We have used Gram–Schmidt process [24, 26, 27] to take advantage of the rank-deficient nature. It is computationally less expensive than SVD. Given an $n \times n$ matrix $\mathbf{A}$ and a user specified tolerance ε, the Gram–Schmidt process computes the numerical rank r of $\mathbf{A}$ and decomposes the matrix into

$$\mathbf{A} = \mathbf{U}\mathbf{V}^{\mathrm{T}} \tag{42}$$

where $\mathbf{U}$ is an $n \times r$ orthogonal matrix which spans the column space of $\mathbf{A}$. The process of multiplying a vector with matrix $\mathbf{A}$ takes $O(2nr)$ operations.

3.2. Procedure

The procedure to rapidly compute the matrix–vector product can be broadly divided into two steps:

1. *Preprocessing*: This step generates the compressed form of the matrix.

   Here, we briefly describe the algorithm for the recursive decomposition of matrix $\mathbf{M}$ generated by the boundary cloud method. Algorithm 1 summarizes the key steps. The basic step is to partition the matrix recursively, as shown in Figure 3, until the size (n) of the submatrix at the lowest level equals a fixed number b.[‡] The recursive partitioning implies that if the size of the submatrix $\mathbf{A}$ is greater than b then recursively construct the representation of the four submatrices $\mathbf{A}_1(1 : \mathrm{size}(\mathbf{A})/2, 1 : \mathrm{size}(\mathbf{A})/2)$, $\mathbf{A}_2(1 : \mathrm{size}(\mathbf{A})/2, (\mathrm{size}(\mathbf{A})/2+1) : \mathrm{size}(\mathbf{A}))$, $\mathbf{A}_3((\mathrm{size}(\mathbf{A})/2+1) : \mathrm{size}(\mathbf{A}), 1 : \mathrm{size}(\mathbf{A})/2)$, $\mathbf{A}_4((\mathrm{size}(\mathbf{A})/2+1) : \mathrm{size}(\mathbf{A}), (\mathrm{size}(\mathbf{A})/2+1) : \mathrm{size}(\mathbf{A})/2)$. The Gram–Schmidt procedure is used to obtain the orthonormal column basis $\mathbf{U}_i$ and rank r_i of the submatrix at the lowest level. If the rank of the child submatrix $\mathbf{A}_i$ is less than $\mathrm{size}(\mathbf{A}_i)/2$, then the submatrix $\mathbf{A}_i$ is stored in the decomposed form. If none of the child submatrices is of low rank, then a UV decomposition of the matrix $\mathbf{A}$ is obtained by the merge procedure given in Algorithm 2. If the rank of $\mathbf{A}$ is less than $\mathrm{size}(\mathbf{A})/2$, then the matrix is stored in decomposed form, otherwise, it is stored in the dense form.

[‡]This number may vary. Matrix of odd size cannot be divided into two equal parts, i.e. the size of child submatrices may be b and $b + b'$, where $b' < b$. Also the value of this number is assumed to lie between $\log N$ and $2 \log N$, where N is the size of the assembled matrix.

Algorithm 1. Constructing a compressed form of the dense matrix.

```
Function Decompose(A, size(A))
if size(A) > b then
  /* recursive subdivision of matrix A into submatrices A1, A2, A3, A4 each of size
  (size(A))/2 */
  A = [A1  A2]
      [A3  A4]
  if size(A1) > b then
    Call decompose(A1, size(A1))
    Call decompose(A2, size(A2))
    Call decompose(A3, size(A3))
    Call decompose(A4, size(A4))
  else
    /* UV decompose Ai of size (size(A))/2 × (size(A))/2, i = 1 to 4 */
    [Ui, Vi, ri] ⇐ Gram–Schmidt decomposition(Ai)
    if ri < (size(Ai))/2 then
      store Ai ⇐ Ui Vi
    else
      /* merge A1, A2, A3, A4 to get UV decomposition of A */
      construct A = [U, V, r] by merging
      if r < (size(A))/2 then
        store A ⇐ UV
      else
        store A
      end if
    end if
  end if
end if
```

In our implementation of the compressed form we do not assemble the entire dense matrix, but compute any (i,j)th term by the boundary cloud method described in the previous section. The required storage space is allocated for submatrices at the lowest level, that is, $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_4$, as shown in Figure 3. Algorithm 1 is then used to get the compressed form of these 4 submatrices and then they are stored in the relevant form (either *UV* form or merged). The same storage space is then used for the next 4 sets of submatrices at that level and this process is carried out till the entire compressed matrix has been constructed.

The process of merging for constructing the *UV* decomposition of submatrix $\mathbf{A}$ of size $n \times n$ is described in Algorithm 2. It consists of 2 horizontal mergings and 1 vertical merging as described below (shown in Figure 4).

(a) Horizontally merge $\mathbf{A}_1$ and $\mathbf{A}_2$, each of size $n/2 \times n/2$: Construct a $n/2 \times (r_1 + r_2)$ matrix $\mathbf{X}$ from the *UV* decompositions of A_1 and A_2, i.e. $\mathbf{X} = [\mathbf{U}_1 \ \mathbf{U}_2]$. Use Gram–Schmidt algorithm to obtain the orthonormal subspace matrix $\mathbf{U}_{12}$ of $\mathbf{X}$ where

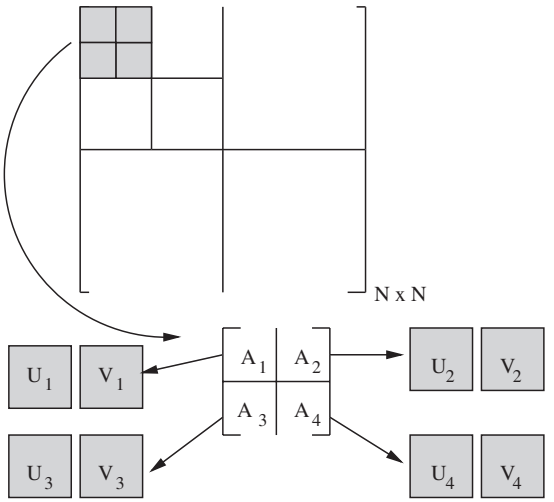

Figure 3. Generation of a compressed form, i.e. recursive division of the submatrices, until the size of the submatrix $n < b$, the small fixed number. Here A_i is of size $n \times n$, U_i is of size $n \times r_i$, V_i is of size $r_i \times n$ and r_i is the rank of the submatrix A_i.

Algorithm 2. Merge algorithm

/* *Horizontally merge* $\mathbf{A}_1$, $\mathbf{A}_2$ */
Construct $\mathbf{X} = [\mathbf{U}_1 \mathbf{U}_2]$
$[\mathbf{U}_X, \mathbf{V}_X, r_{12}] =$ Gram–Schmidt decomposition($\mathbf{X}$)
$\mathbf{U}_{12} \Leftarrow \mathbf{U}_X$
$\mathbf{V}_{12} \Leftarrow \mathbf{U}_{12}^T [\mathbf{U}_1 \mathbf{V}_1 \mathbf{U}_2 \mathbf{V}_2]$
/* *Horizontally merge* $\mathbf{A}_3$, $\mathbf{A}_4$ */
Construct $\mathbf{Y} = [\mathbf{U}_3 \mathbf{U}_4]$
$[\mathbf{U}_Y, \mathbf{V}_Y, r_{34}] =$ Gram–Schmidt decomposition($\mathbf{Y}$)
$\mathbf{U}_{34} \Leftarrow \mathbf{U}_Y$
$\mathbf{V}_{34} \Leftarrow \mathbf{U}_{34}^T [\mathbf{U}_3 \mathbf{V}_3 \mathbf{U}_4 \mathbf{V}_4]$
/* *Vertically merge* $\mathbf{A}_{12}$, $\mathbf{A}_{34}$ */
Construct $\mathbf{Z} = [\mathbf{V}_{12}^T \mathbf{V}_{34}^T]$
$[\mathbf{U}_Z, \mathbf{V}_Z, r] =$ Gram–Schmidt decomposition($\mathbf{Z}$)
$\mathbf{V} \Leftarrow \mathbf{U}_Z^T$

$$\mathbf{U} \Leftarrow \begin{bmatrix} \mathbf{U}_{12}\mathbf{V}_{12} \\ \mathbf{U}_{34}\mathbf{V}_{34} \end{bmatrix} \mathbf{V}^T$$

r_{12} is the rank of $\mathbf{X}$. Then compute the $r_{12} \times n/2$ matrix V_{12} as given in Algorithm 2. Now, $[\mathbf{A}_1 \ \mathbf{A}_2] = \mathbf{U}_{12}\mathbf{V}_{12} = \mathbf{A}_{12}$.

(b) Horizontally merge $\mathbf{A}_3$ and $\mathbf{A}_4$, each of size $n/2 \times n/2$: Construct a $n/2 \times (r_3 + r_4)$ matrix $\mathbf{Y}$ from the UV decompositions of A_3 and A_4, i.e. $\mathbf{Y} = [\mathbf{U}_3 \ \mathbf{U}_4]$. Use

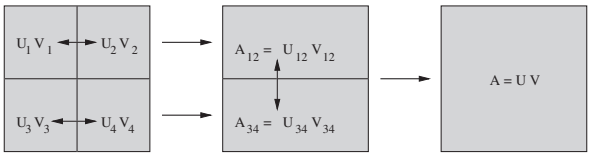

Figure 4. If none of the submatrices $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_4$ is of low rank, a merging process is performed to obtain a UV decomposition of matrix $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{bmatrix}$. The objective of the merging process is to compute the UV decomposition of matrix $\mathbf{A}$ using the UV decomposition of each of its submatrices.

Algorithm 3. Matrix-Vector Product

if $\mathbf{a} = \mathbf{UV}$ **then**
  compute $y = \mathbf{U}(\mathbf{V}x)$
else
  compute $y = \mathbf{a}x$
end if

Gram–Schmidt algorithm to obtain the orthonormal subspace matrix $\mathbf{U_{34}}$ of $\mathbf{Y}$ where r_{34} is the rank of $\mathbf{Y}$. Then compute the $r_{34} \times n/2$ matrix V_{34} as given in Algorithm 2. Now, $[\mathbf{A}_3 \ \mathbf{A}_4] = \mathbf{U}_{34}\mathbf{V}_{34} = \mathbf{A}_{34}$.

(c) Vertically merge $\mathbf{A}_{12}$ and $\mathbf{A}_{34}$, each of size $n/2 \times n$: Construct a $n \times (r_{12} + r_{34})$ matrix $\mathbf{Z}$ as given in Algorithm 2. Use Gram–Schmidt algorithm to obtain the orthonormal $n \times r$ subspace matrix $\mathbf{V}^{\mathrm{T}}$ of $\mathbf{Z}$ where r is the rank of $\mathbf{Z}$. Finally, compute the $n \times r$ matrix $\mathbf{U}$ as given in Algorithm 2. Now, $\mathbf{A} = \mathbf{UV}$.

The complexity of preprocessing is dependent on computing the UV decompositions of the submatrices and is equivalent to computing 4 or 5 dense matrix–vector products.

2. *Solve the linear system using* GMRES: The compressed representation is then used to compute the matrix–vector products in an iterative solver (like GMRES). The matrix–vector multiplication is performed by the following recursive algorithm:
Let $\mathbf{a}$ be the current submatrix of size $n \times n$, x be the vector in the ith iteration of GMRES and y be the output vector obtained in the ith iteration. As explained in Algorithm 3 and shown in Figure 5, if the submatrix $\mathbf{a}$ is represented as $\mathbf{UV}$, then compute $y = \mathbf{U}(\mathbf{V}x)$, else perform the dense matrix vector product.

In summary, the acceleration technique described above requires a preprocessing step which costs about 4 or 5 dense matrix–vector products. The storage requirement is at most $O(N(\log N)^2)$ and the dense matrix–vector product can be computed in at most $O(N(\log N)^2)$ operations using the compressed matrix generated by the preprocessing approach (see Appendix A for a proof on the complexity of the approach).

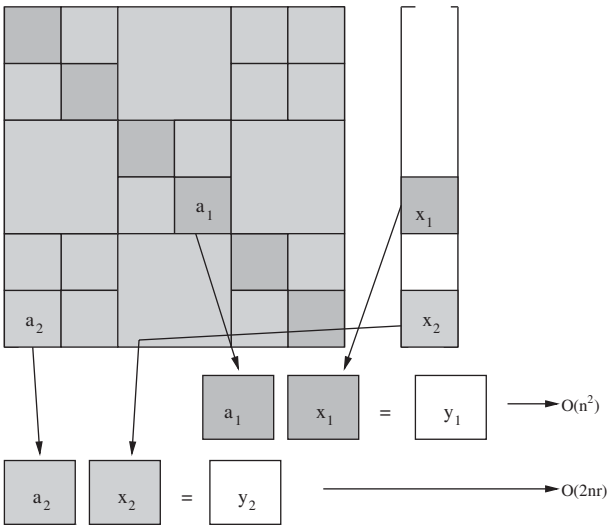

Figure 5. Calculation of matrix–vector product. A shaded or a dark region indicates that the submatrix is stored in the dense form. A non-shaded region indicates that the submatrix is stored in the *UV* form. n is the size of submatrix a_1 or a_2 and r is the rank of submatrix a_2.

4. RESULTS

In this section we present results for three problems, solved by applying the fast BCM described in the previous section. For all the examples the numerical precision ε for computing the rank is 1×10^{-6}.

4.1. Two-conductor problem

The configuration of the two-conductor example is shown in Figure 6. BCM has been used to compute the charge density along the boundary of the two conductors. The problem has been solved by using different number of nodes, i.e. 16, 32, 64 and 128 per edge per conductor. The fixed number b varies with different number of nodes, ranging from 16 to 32.

Figure 7 shows the plot of the assembled compressed matrix, when 128 nodes were used per edge for each conductor. It shows that large portions of the matrix are rank deficient (numbers in the boxes indicate the ranks of the submatrices). Figures 8 and 9 show how the storage and the number of floating point operations scale with the number of nodes. We observe that both of them scale as $O(N(\log N)^2)$. The number of floating point operations in the matrix–vector product are directly related to the number of elements stored in the matrix and this explains their same order. The result in Figure 9 indicates that by using the SVD-based acceleration technique, the dense matrix–vector product for a two-conductor problem can be reduced to $O(N(\log N)^2)$ operations instead of the classical $O(N^2)$ operations. Detailed analysis of storage and matrix–vector product operations is explained in Appendix A.

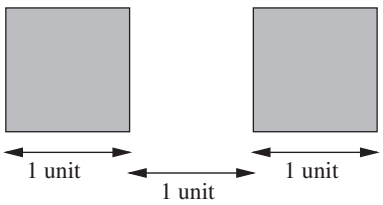

Figure 6. Configuration of the two-conductor problem.

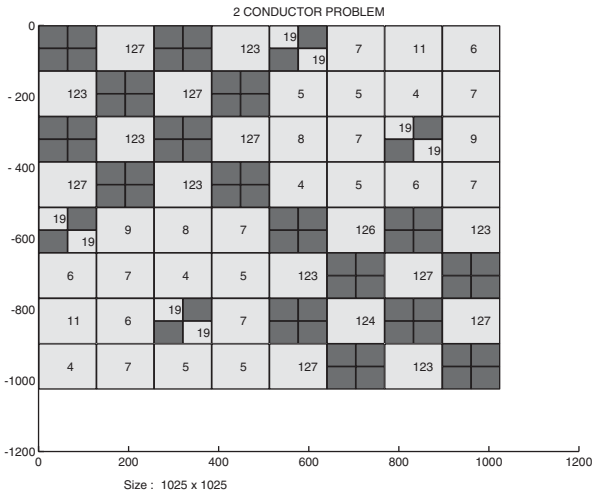

Figure 7. Assembled matrix for the two-conductor problem. A shaded or a dark region denotes that the submatrix is stored in the dense form, while the non-shaded region denotes that the matrix is stored in the *UV* decomposed form with the numerical value of rank on it.

4.2. Mirror problem

Micromirrors are used as spatial light modulators in high performance display applications. The micromirror is actuated by an electrostatic force which is generated by applying a voltage difference between the mirror plate and the electrode. Figure 10 shows the geometry of the mirror example. The configuration is almost identical to that of the two-conductor example, except for the aspect ratio.

The problem has been solved by using different number of nodes, i.e. 40, 80, 160 and 320 nodes per conductor. The fixed number b varies with different number of nodes, ranging from 10 to 40. Figure 11 shows the plot of the assembled matrix, when 160 nodes were used per conductor. It shows that many portions of the dense matrix have low rank (numbers in the boxes are the ranks of the submatrices) and the rank deficiency can be exploited to accelerate the solution process and to reduce the storage space. Figure 12 and Figure 13 depict the scaling of the storage space and the number of floating point operations with different

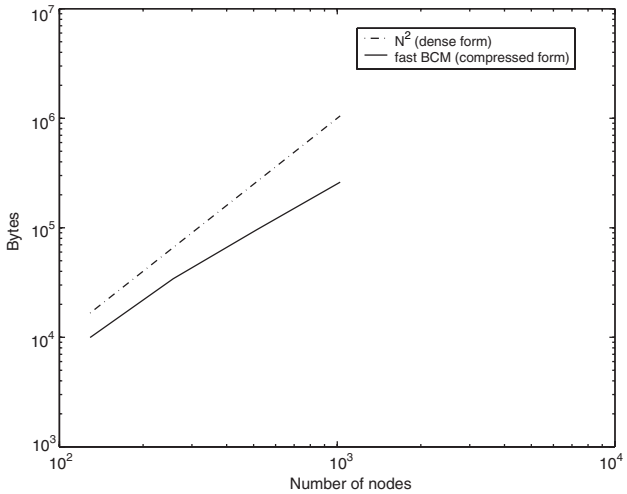

Figure 8. Scaling of storage with the number of nodes for the two-conductor example. The solid line shows the storage space occupied by the compressed matrix, while the dash–dot line shows that required for the storage of the dense form.

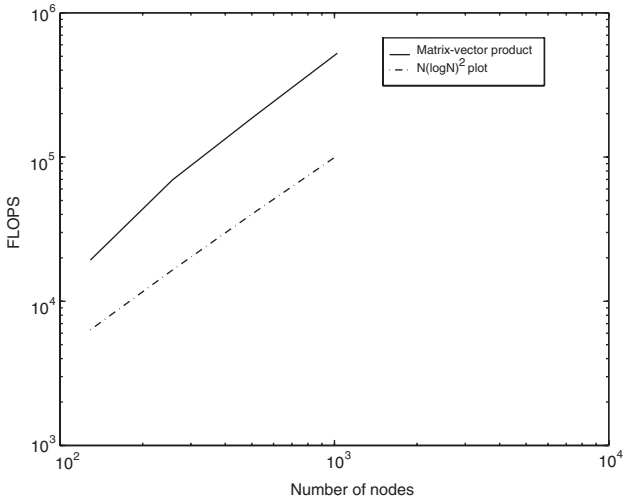

Figure 9. Scaling of flops with the number of nodes for the two-conductor problem. The solid line shows flops for computing the matrix–vector product in GMRES with the compressed form and it scales same as the dash–dot line plot, i.e. $O(N(\log N)^2)$.

number of nodes. We observe that both of them scale as $O(N(\log N)^2)$. The explanation for the scaling of the storage and the computation of the matrix–vector product is given in Appendix A.

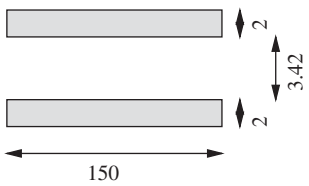

Figure 10. Configuration of the mirror problem.

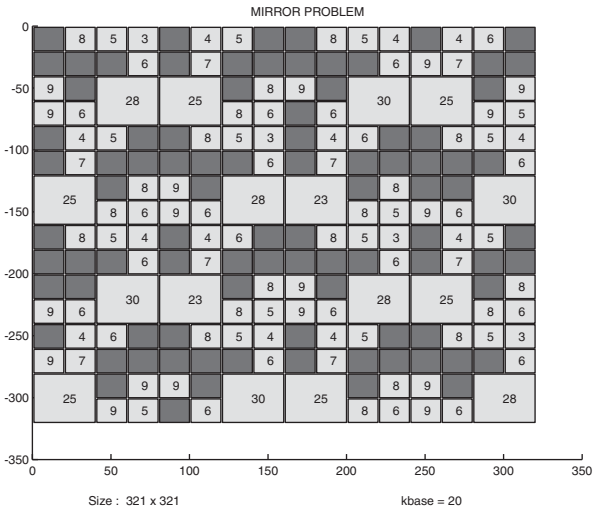

Figure 11. Assembled matrix for the mirror problem. A shaded or dark region denotes that the submatrix is stored in the dense form, while the non-shaded region denotes that the matrix is stored in the *UV* decomposed form with the numerical value of rank on it.

4.3. Comb-drive problem

Comb drive is a common MEMS actuator device which consists of rows of interlocking teeth; half of the teeth are attached to a fixed beam and the other half attached to a movable beam assembly. A section of a comb drive has been solved. The configuration is shown in Figure 14. We observe that there are more number of points that are quite far-away compared to the number of points that are near by. Hence, we could look into the effect of smoothness of the Green's function when the points are quite far away.

The problem has been solved for varying number of nodes. The fixed number b varies with the number of nodes and ranges from 16 to 30. Figure 15 shows the assembled matrix of size 415×415. The numbers in the boxes indicate the rank of the submatrices. Figures 16 and 17 show the scaling of the storage space used and the number of floating point operations with different number of nodes. Both of them scale as $O(N \log N)$. The lower order of complexity for the storage and the computation of matrix–vector products in the comb-drive problem, as compared to the two-conductor and the mirror problems, is because the dense matrix contains fewer near-by points and more far-away interactions (see Appendix A for more details).

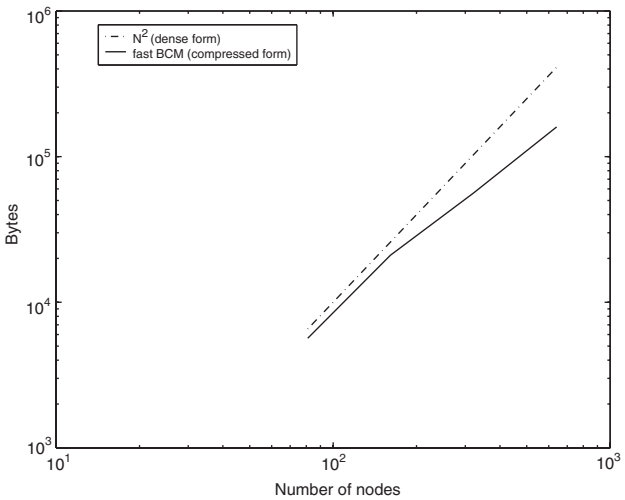

Figure 12. Scaling of storage with the number of nodes for the mirror example. The solid line shows the storage space occupied by the compressed matrix, while the dash–dot line shows that required for the storage of the dense form.

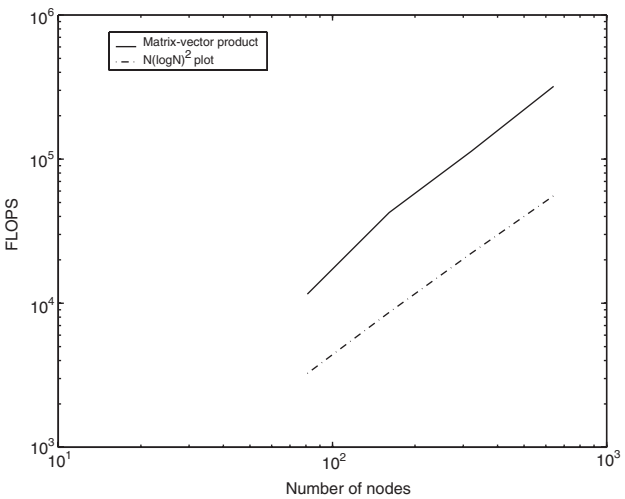

Figure 13. Scaling of flops with the number of nodes for the mirror problem. The solid line shows flops for computing the matrix–vector product in GMRES with the compressed form and it scales same as the dash–dot line plot, i.e. $O(N(\log N)^2)$.

5. CONCLUSION

A fast boundary cloud method for numerical solution of exterior two-dimensional electrostatic problems has been presented in this paper. The acceleration technique is based on the idea of

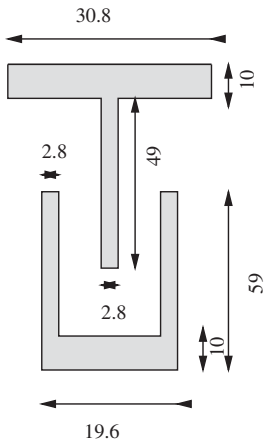

Figure 14. Configuration of the comb-drive problem.

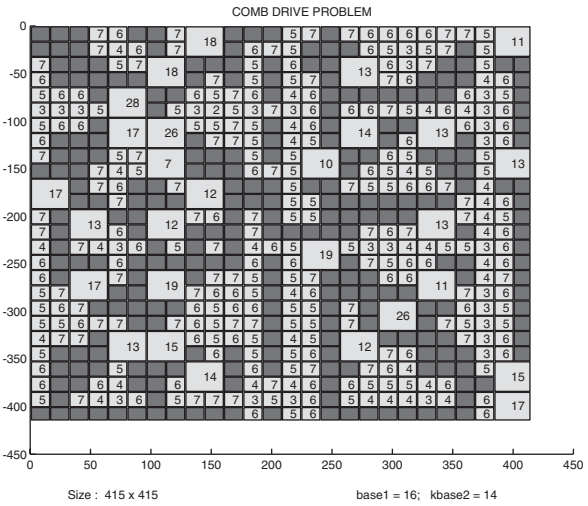

Figure 15. Assembled matrix for the comb-drive problem. A shaded or dark region denotes that the submatrix is stored in the dense form, while the non-shaded region denotes that the matrix is stored in the *UV* decomposed form with the numerical value of the rank on it.

recursively subdividing the dense matrix and storing the submatrix either in a *UV* decomposed form or in a dense form. The algorithm has been applied to three different configurations and from the results obtained we can conclude that the memory used and the solution time scale identically for the two conductor and the mirror problem as $O(N(\log N)^2)$ and for the comb-drive problem as $O(N \log N)$. The results indicate that when the dense matrix involves predominantly near-by interactions the complexity is $O(N(\log N)^2)$, otherwise, the complexity is $O(N \log N)$.

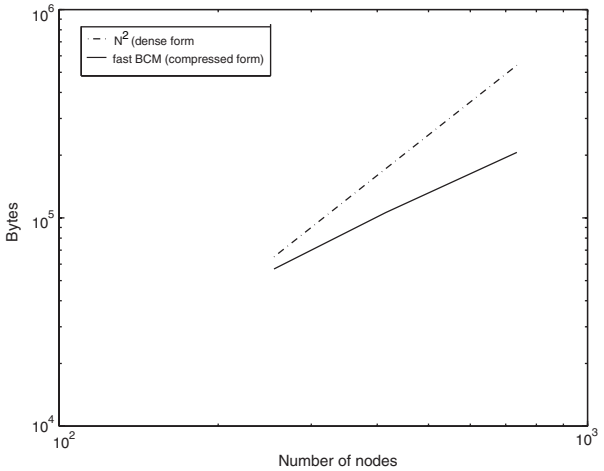

Figure 16. Scaling of storage with the number of nodes for the comb-drive example. The solid line shows the storage space occupied by the compressed matrix, while the dash–dot line shows that required for the storage of the dense form.

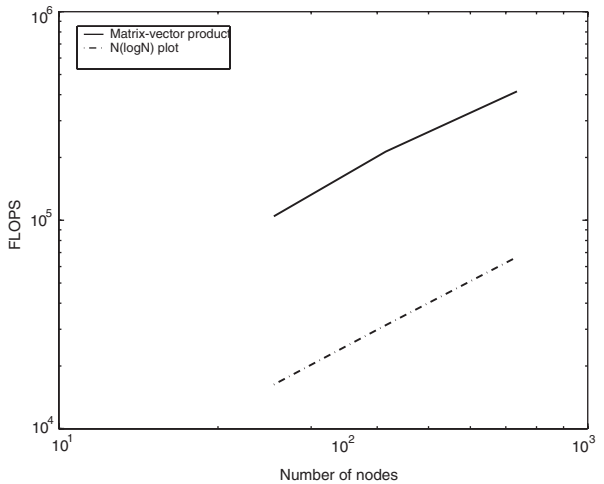

Figure 17. Scaling of flops with the number of nodes for the comb-drive problem. The solid line shows flops for computing the matrix–vector product in GMRES with the compressed form and it scales same as the dash–dot line plot, i.e. $O(N \log N)$.

APPENDIX A

A.1. 3D versus 2D

The complexity and storage analysis of the acceleration technique is discussed in detail below. Figure A1 shows a comparison between the three-dimensional kernel or Green's function $(1/r)$

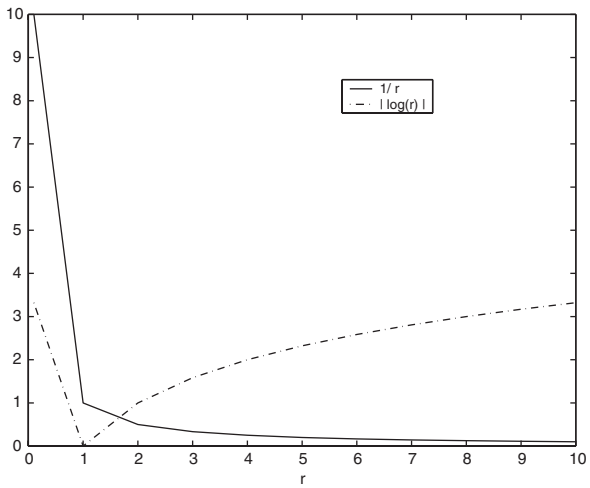

Figure A1. Plot of $|\log(r)|$ and $1/r$. Both of them are asymptotically smooth but $|\log(r)|$ has higher gradient as compared to $1/r$ plot. This results in a higher difference in function value for two near-by points in case of $|\log(r)|$ as compared to $1/r$.

and the two-dimensional kernel or Green's function ($|\log(r)|$). The acceleration technique exploits the fact that the difference in the value of the Green's function is very small for neighbouring source points when they are far away from the field point. From the plot it is evident that this difference is much smaller in the case of $1/r$ kernel as compared to that in $|\log(r)|$. This explains the difference in the complexity of three- and two-dimensional problems.

A.2. Complexity analysis

The complexity of computing matrix–vector product with the compressed form depends on determining the number of operations required in multiplying the submatrices with the vector. The submatrices lying along the main diagonal (Level 0 in Figure A2) depict the interaction between near-by source and field points. As we move away from the main diagonal (Level 1, Level 2,..., Level ($\log N$) in Figure A2), the submatrices are a result of interaction between farther away source and field points and the rank-deficient behaviour becomes more and more evident. The size and the number of submatrices along each diagonal are estimated and the number of operations required to multiply the submatrix with a vector is computed. The number of operations performed along each diagonal are summed up to give the total.

The complexity analysis can be divided into three steps:

1. b, a small fixed number, is assumed to take a value between $\log N$ and $2\log N$, where N is the size of the assembled matrix. Therefore, b is of $O(\log N)$.

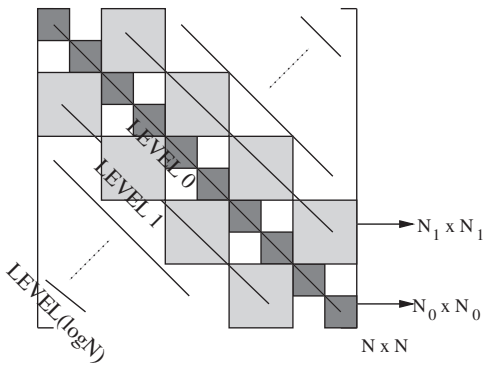

Figure A2. The plot shows the different levels, i.e. subdiagonals in the assembled matrix of size $N \times N$ and the submatrices along each of them.

2. Basic idea of the acceleration technique is to express a matrix of size $N_1 \times N_1$ as a product of 2 matrices, i.e.

$$\mathbf{A} = \mathbf{U}\mathbf{V}^{\mathrm{T}} \tag{A1}$$

where **U** and **V** are of size $N_1 \times r$ and r is the rank of the matrix **A**. When a matrix **A** is expressed in the **UV** form, the number of operations in multiplying the matrix with a vector are $O(N_1 r)$.

Consider the following two cases to estimate the order of r:

(a) *Case A*: Matrix **A** involves interaction between near-by points. In this case, $r < b/2$. Therefore, r is $O(\log N)$. In this case the matrix–vector product operations is

$$O(N_1 r) = O(N_1 \log N) \tag{A2}$$

(b) *Case B*: Matrix **A** involves interaction between far-away points. In this case, $r \ll b/2$. Therefore, r is $O(1)$ and the order of multiplying a matrix **A** with a vector is

$$O(N_1 r) = O(N_1) \tag{A3}$$

3. (a) *Case A*: The entire matrix involves predominantly nearby interactions (e.g. the case of two-conductor problem and the mirror problem, presented in the results section). Level 0 (submatrices along the main diagonal, see Figure A2):

Size of submatrices $= N_0$

Number of submatrices $= \tilde{N}_0 = N/N_0$

Multiplication of $\tilde{N}_0$ submatrices with a vector is of order $= \tilde{N}_0(N_0 \log N)$

$= (N/N_0)N_0 \log N$

$= N \log N$

Level 1 (submatrices along the subdiagonals next to the main diagonal):

Size of submatrices $= N_1$

Number of submatrices $= \tilde{N}_1 = N/N_1$

Multiplication of $\tilde{N}_1$ submatrices with a vector is of order $= \tilde{N}_1(N_1 \log N)$

$$= (N/N_1)N_1 \log N$$

$$= N \log N$$

$\vdots$

Number of levels (i.e. number of subdiagonals) $= \log N$.

Summing up we get

$$\underbrace{(N \log N + N \log N + \cdots + N \log N)}_{\log N \ \ times} = (N \log N) \log N = N(\log N)^2 \tag{A4}$$

(b) Case B: The entire matrix involves predominantly far-away interactions, as in the case of comb-drive problem.

Level 0 (submatrices along the main diagonal, see Figure A2):

Size of submatrices $= N_0$

Number of submatrices $= \tilde{N}_0 = N/N_0$

Multiplication of $\tilde{N}_0$ submatrices with a vector is of order $= \tilde{N}_0 N_0$

$$= (N/N_0)N_0$$

$$= N$$

Level 1 (submatrices along the subdiagonals next to the main diagonal):

Size of submatrices $= N_1$

Number of submatrices $= \tilde{N}_1 = N/N_1$

Multiplication of $\tilde{N}_1$ submatrices with a vector is of order $= \tilde{N}_1 N_1$

$$= (N/N_1)N_1$$

$$= N$$

$\vdots$

Number of levels (i.e. number of subdiagonals) $= \log N$.

Summing up we get

$$\underbrace{(N + N + \cdots + N)}_{\log N \ \ times} = N \log N \tag{A5}$$

A.3. Storage analysis

The storage analysis is the same as the analysis for computation of matrix–vector products. The storage space required for the *UV* decomposed form is of the same order as that of computing a matrix–vector product with the compressed form.

ACKNOWLEDGEMENTS

The authors would like to thank Gang Li for his help with the implementation of the boundary cloud method. This work is supported by an NSF CAREER award to N.R. Aluru.

REFERENCES

1. Aluru NR, White J. An efficient numerical technique for electromechanical simulation of complicated micro-electro-mechanical structures. *Sensors and Actuators A* 1997; **58**:1–11.
2. Aluru NR, White J. A multilevel Newton method for mixed-energy domain simulation of MEMS. *Journal of Microelectromechanical Systems* 1999; **8**(3):299–308.
3. Gang Li, Aluru NR. Linear, nonlinear and mixed-regime analysis of electrostatic MEMS. *Sensors and Actuators A* 2001; **91**:278–291.
4. Senturia SD, Aluru NR, White J. Simulating the behavior of MEMS devices: computational methods and needs. *IEEE Computational Science and Engineering* 1997; **4**(1):30–43.
5. Shi F, Ramesh P, Mukherjee S. On the application of 2D potential theory to electrostatic simulation. *Communications in Numerical Methods in Engineering* 1995; **11**:691–701.
6. Kane JH. *Boundary Element Analysis in Engineering Continuum Mechanics*. Prentice-Hall: Englewood Cliffs, NJ, 1994.
7. Aluru NR, Gang Li. Finite cloud method: a true meshless technique based on a fixed reproducing kernel approximation. *International Journal for Numerical Methods in Engineering* 2001; **50**:2373–2410.
8. Atluri SN, Zhu T. A new meshless local Petrov–Galerkin (MLPG) approach in computational mechanics. *Computational Mechanics* 1998; **22**:117–127.
9. Belytschko T, Krongauz Y, Organ D, Flemming M, Krysl P. Meshless methods: an overview and recent developments. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**:3–47.
10. Belytschko T, Lu YY, Gu L. Element free Galerkin methods. *International Journal for Numerical Methods in Engineering*. 1994; **37**:229–256.
11. Duarte CA, Oden JT. An h-p adaptive method using clouds. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**:237–262.
12. Liu WK, Jun S, Zhang YF. Reproducing kernel particle methods. *International Journal for Numerical Methods in Fluids* 1995; **20**:1081–1106.
13. Lizka TJ, Duarte CAM, Tworzydlo WW. hp-Meshless cloud method. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**:263–288.
14. Melenk JM, Babuska I. The partition of unity finite element method: basic theory and applications. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**:289–314.
15. Monaghan JJ. An introduction to SPH. *Computer Physics Communications* 1988; **48**:89–96.
16. Oñate E, Idelsohn S, Zienkiewicz OC, Taylor RL. A finite point method in computational mechanics. Applications to convective transport and fluid flow. *International Journal for Numerical Methods in Engineering* 1996; **39**:3839–3866.
17. Gu YT, Liu GR. A boundary point interpolation method for stress analysis of solids. *Computational Mechanics* 2002; **28**:47–54.
18. Mukherjee YX, Mukherjee S. The boundary node method for potential problems. *International Journal for Numerical Methods in Engineering* 1997; **40**(5):797–815.
19. Gang Li, Aluru NR. Boundary cloud method: a combined scattered point/boundary integral approach for boundary-only analysis. *Computer Methods in Applied Mechanics and Engineering*, 2002; **191**:2337–2370.
20. Greengard L, Rokhlin V. A fast algorithm for particle simulations. *Journal of Computational Physics* 1987; **73**(2):325–348.
21. Phillips JR, White JK. A precorrected-FFT method for electrostatic analysis of complicated 3-D structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 1997; **16**(10):1059–1072.
22. Kapur S, Long DE. *IES*^{3}: a fast integral equation solver for efficient 3-dimensional extraction. *IEEE Computer Aided Design*, 1997, *Digest of Technical Papers 1997*, *IEEE/ACM International Conference*, 1997; 448–455.
23. Kapur S, Zhao J. A fast method of moments solver for efficient parameter extraction of MCMs. *Design Automation Conference*, 1997, *Proceedings of the 34th Conference*, 1997;141–146.
24. Heath MT. *Scientific Computing—An Introductory Survey*. McGraw-Hill: New York, 1997.
25. Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**(3):856–869.
26. Barrett B, Berry M, Chan TF, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C, Vorst H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM: Philadelphia, PA, 1994.
27. Trefethen LN, Bau D. *Numerical Linear Algebra. SIAM*: Philadelphia, PA, 1997.

