FMM/GPU-Accelerated Boundary Element Method for Computational Magnetics and Electrostatics

Ross Adelman, Nail A. Gumerov, and Ramani Duraiswami

***Abstract*—A fast multipole method (FMM)/graphics processing unit (GPU)-accelerated boundary element method (BEM) for computational magnetics and electrostatics via the Laplace equation is presented. The BEM is an integral method, but the FMM is typically designed around monopole and dipole sources. To apply the FMM to the integral expressions in the BEM, the internal data structures and logic of the FMM must be changed. However, this can be difficult. For example, computing the multipole expansions due to the boundary elements requires computing single and double surface integrals over them. Moreover, FMM codes for monopole and dipole sources are widely available and highly optimized. This paper describes a method for applying the FMM unchanged to the integral expressions in the BEM. This method, called the correction factor matrix method, works by approximating the integrals using a quadrature. The quadrature points are treated as monopole and dipole sources, which can be plugged directly into current FMM codes. The FMM is effectively treated as a black box. Inaccuracies from the quadrature are corrected during a correction factor step. The method is derived, and example problems are presented showing accuracy and performance.**

***Index Terms*—Boundary element method, boundary integral equations, fast solvers, Galerkin method, integral equations, Laplace equation, method of moments, parallel processing, parallel programming**

I. Introduction

The boundary element method (BEM), also known as the method of moments, is a powerful method for solving many partial differential equations (PDEs), including the Laplace equation [1]–[3]. The BEM has a number of advantages, including that it reduces the space dimensionality of the discretization by one, handles the boundary conditions at infinity accurately, and treats complex boundary shapes and multi-domain problems well. As a result, it is used extensively in electromagnetism [4], plasma physics [5], astrophysics [6], aerodynamics [7], [8], and for computing the values in lumped-parameter models for circuit design and analysis [9], [10]. In this paper, we focus primarily on problems in the domains of magnetics and electrostatics. The BEM suffers from a few problems. The linear systems arising in the BEM are conventionally solved via direct matrix decompositions. However, the system matrices are dense, requiring $\mathrm{O}\left(N^2\right)$ storage, where N is the number of discretization unknowns. This can effectively restrict the size of a problem that can be solved on a given machine. For example, on a machine with 8 GB of main memory, problem sizes are restricted to around 30,000 boundary elements, assuming double precision is used. Moreover, these decompositions require $\mathrm{O}\left(N^3\right)$ operations. Using an iterative solver based on Krylov subspace methods, such as GMRES [11], alleviates this issue somewhat, providing an $\mathrm{O}\left(N_{\text{iter}}N^2\right)$ method, where N_{iter} is the number of iterations and $\mathrm{O}\left(N^2\right)$ is the cost of the matrix-vector product that is computed during each iteration. However, for large N, this quadratic scaling in both storage and time can still be prohibitive.

The fast multipole method (FMM) allows for the acceleration of certain matrix-vector products, and was originally developed to address the N-body problem arising in stellar and molecular dynamics [12], [13]. The FMM computes a matrix-vector product to any specified error, ε, in $\mathrm{O}\left(N\right)$ storage and time, where the asymptotic constant depends on ε. There has been significant effort to speed up the FMM by parallelizing across many cores using OpenMP, on the graphics processing unit (GPU) using CUDA [14], or across an entire cluster [15], [16]. Some open source FMM codes are available [17]. Here, we used our own highly optimized, in-house, GPU-accelerated FMM.

The FMM is usually designed around the summing of monopole and dipole sources, and not the integral expressions that appear in the BEM. Extending the FMM to handle these integral expressions is possible, but requires modifying the internal data structures and logic. Many authors have done so. For example, the FMM was used to solve problems in elastostatics [18], [19], problems in acoustics and the computation of head-related transfer functions [20], [21], and electromagnetic scattering problems [22]. As seen in these references, the issues associated with adding the integral expressions directly to the FMM are manageable, and the results are good, but these changes complicate the inner workings, or might compromise the accuracy of the BEM implementation. Moreover, FMM codes for monopole and dipole sources are widely available and highly optimized. Newly developed FMM codes for integral-based methods may not be as highly optimized. Thus, we seek a procedure for applying efficient FMM codes unchanged to the integral expressions in the BEM.

This approach, which we call the correction factor matrix method, is described in this paper. The method approximates the integrals using a possibly inaccurate quadrature, and then treats the quadrature points as monopole and dipole sources, which can be plugged directly into the FMM. Any inaccuracies from the quadrature are corrected during a correction factor step, which runs in $\mathrm{O}\left(N\right)$. The FMM is effectively treated as a black box, and the correction factor matrix method is

Ross Adelman is with the Army Research Laboratory, Adelphi, MD 20783 (email: ross.n.adelman.civ@mail.mil)

Nail A. Gumerov is with the University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742, and Fantalgo, LLC, Elkridge, MD 21075 (email: gumerov@umiacs.umd.edu)

Ramani Duraiswami is with UMIACS, the University of Maryland Department of Computer Science, College Park, MD 20742, and Fantalgo, LLC (email: ramani@umiacs.umd.edu)

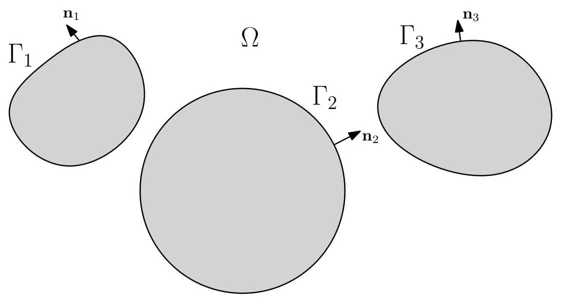

Fig. 1. The boundary can be arbitrarily shaped, even made of several disconnected boundaries, but they must all be closed.

simply a series of pre- and post-processing steps, which also run in $\mathrm{O}(N)$ and achieve the prescribed error tolerance. We also accelerate this method using the GPU.

Similar point-based approaches have been proposed in the past for solving radiation problems [23], [24]. We build on this previous work in several ways. First, we apply the method to the Laplace equation. Second, we treat the FMM as a black box. We only rely on the FMM to compute potentials and gradients due to monopole and dipole sources. In fact, any fast matrix-vector product algorithm could be used. Third, we explore how varying the different available parameters affects the accuracy and performance of the method. We study this in the context of an example magnetics problem (a perfectly conducting sphere in a uniform magnetic field) and an example electrostatics problem (an electric-field cage). The correction factor matrix method is shown to scale linearly, be able to solve problems with a million elements in only minutes on a single workstation, and yield errors below 0.01%.

II. Background

The electric and magnetic fields in an environment are completely described by Maxwell's equations [25]. In the quasistatic approximation, the electric and magnetic fields change slowly enough in time that their time derivatives can be ignored. This causes the equations for the electric and magnetic fields to decouple, leaving two sets of equations, one for magnetoquasistatics (MQS) problems and the other for electroquasistatics (EQS) problems. The MQS equations are:

$$\nabla\cdot\mathbf{B}=0,\quad \nabla\times\mathbf{H}=\mathbf{J}. \tag{1}$$

The EQS equations are:

$$\nabla\cdot\mathbf{D}=\rho,\quad \nabla\times\mathbf{E}=\mathbf{0}. \tag{2}$$

In the absence of free charge or current in the domain ($\rho=0$ and $\mathbf{J}=\mathbf{0}$), and assuming linear materials ($\mathbf{D}=\varepsilon\mathbf{E}$ and $\mathbf{B}=\mu\mathbf{H}$), these two sets of equations each reduce to the Laplace equation:

$$\nabla^2\phi^{\mathrm{M}}=0,\quad \mathbf{H}=-\nabla\phi^{\mathrm{M}}, \tag{3}$$

$$\nabla^2\phi^{\mathrm{E}}=0,\quad \mathbf{E}=-\nabla\phi^{\mathrm{E}}. \tag{4}$$

Solving an MQS or EQS problem, therefore, involves solving the following boundary vale problem (BVP) for the Laplace equation:

$$\nabla^2\phi(\mathbf{x})=0,\quad \mathbf{x}\in\Omega, \tag{5}$$

$$\alpha(\mathbf{x})\,\phi(\mathbf{x})+\beta(\mathbf{x})\,q(\mathbf{x})=\gamma(\mathbf{x}),\quad \mathbf{x}\in\Gamma, \tag{6}$$

where ϕ is either ϕ^{M} or ϕ^{E},

$$q(\mathbf{x})=\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{x})=(\mathbf{n}\cdot\nabla_{\mathbf{x}})\,\phi(\mathbf{x}) \tag{7}$$

is the normal derivative on the boundary, and the boundary conditions have been set appropriately. The boundary can be arbitrarily shaped, even made of several disconnected boundaries, but they must all be closed (see Fig. 1). For external problems, the potential should also decay to zero at large distances:

$$\lim_{|\mathbf{x}|\to\infty}\phi(\mathbf{x})=0. \tag{8}$$

To solve the BVP, we use the direct boundary integral formulation. Using Green's identity, the Laplace equation is transformed from a PDE into an integral equation:

$$\pm\phi(\mathbf{x})=L[q](\mathbf{x})-M[\phi](\mathbf{x}),\quad \mathbf{x}\notin\Gamma, \tag{9}$$

where

$$L[q](\mathbf{x})=\int_{\mathbf{x}'\in\Gamma}q(\mathbf{x}')\,G(\mathbf{x}-\mathbf{x}')\,dS(\mathbf{x}'), \tag{10}$$

$$M[\phi](\mathbf{x})=\int_{\mathbf{x}'\in\Gamma}\phi(\mathbf{x}')\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}G(\mathbf{x}-\mathbf{x}')\right)dS(\mathbf{x}') \tag{11}$$

are the single- and double-layer potentials [26], respectively, and $G(\mathbf{r})=1/(4\pi|\mathbf{r}|)$. The $\pm$ in Eq. (9) should be a $+$ for internal problems and a $-$ for external problems. Due to the behavior of the double-layer potential, for points on the boundary,

$$\pm\frac{1}{2}\phi(\mathbf{x})=L[q](\mathbf{x})-M[\phi](\mathbf{x}),\quad \mathbf{x}\in\Gamma. \tag{12}$$

In the PDE, we seek a solution to the potential in the entire domain. In the integral equation, we seek a solution to the potential and normal derivative only on the boundary. Green's identity can then be used to compute the solution at any point in the domain.

The boundary is discretized using planar triangular elements, and the integrals over the original boundary become sums of integrals over these triangles. The potential, $\phi(\mathbf{x})$, and normal derivative, $q(\mathbf{x})$, on the triangles are each approximated as a linear combination of N basis functions:

$$\phi(\mathbf{x})=\sum_{j=1}^{N}\phi_j f_j(\mathbf{x}),\quad q(\mathbf{x})=\sum_{j=1}^{N}q_j f_j(\mathbf{x}). \tag{13}$$

To get a solution, we need to compute the coefficients, ϕ_j and q_j, so that the boundary conditions are satisfied. The integral equations become

$$\sum_{j=1}^{N}\phi_j\left(M[f_j]\pm\frac{1}{2}f_j\right)-\sum_{j=1}^{N}q_jL[f_j]=0, \tag{14}$$

$$\sum_{j=1}^{N}\phi_j\alpha f_j+\sum_{j=1}^{N}q_j\beta f_j=\gamma. \tag{15}$$

The discretization introduces geometric and approximation errors, which are discussed further in Sec. V-B.

Two common methods for enforcing the boundary conditions are the collocation and Galerkin methods. The collocation method works by enforcing the boundary conditions at N matching points [27], [28]. The integral expressions necessary for implementing the collocation method have been derived for piecewise constant and linear basis functions on triangular elements by many authors [29]–[33]. However, many of the boundary integrals are hypersingular, which make them difficult to compute, especially for points on the corners or edges of the boundary. Furthermore, the resulting system matrices are non-symmetric. The Galerkin method enforces the boundary conditions in an integral sense. The boundary integral equations are multiplied by each of the N basis functions and integrated over the boundary a second time. By doing so, all of the hypersingular integrals become weakly singular. Moreover, the system matrices arising in the Galerkin method are typically symmetric, better conditioned, and have better convergence properties [34], [35]. However, the extra integral over the boundary complicates the computation of the entries in the system matrix. In both methods, the system matrix takes the following form:

$$\mathrm{S} = \begin{bmatrix} \mathrm{U}+\mathrm{D} & \mathrm{V} \\ \mathrm{A} & \mathrm{B} \end{bmatrix}, \tag{16}$$

where U, V, D, A, and B are $N \times N$ matrices. The (i, j)th entries of these matrices are given by

$$U_{i,j}^{(\mathrm{col})} = M\left[f_j\right](\mathbf{x}_i), \quad U_{i,j}^{(\mathrm{gal})} = \int_\Gamma f_i M\left[f_j\right] dS, \tag{17}$$

$$V_{i,j}^{(\mathrm{col})} = L\left[f_j\right](\mathbf{x}_i), \quad V_{i,j}^{(\mathrm{gal})} = \int_\Gamma f_i L\left[f_j\right] dS, \tag{18}$$

$$D_{i,j}^{(\mathrm{col})} = \pm\frac{1}{2} f_j(\mathbf{x}_i), \quad D_{i,j}^{(\mathrm{gal})} = \pm\frac{1}{2}\int_\Gamma f_i f_j dS, \tag{19}$$

$$A_{i,j}^{(\mathrm{col})} = \alpha(\mathbf{x}_i) f_j(\mathbf{x}_i), \quad A_{i,j}^{(\mathrm{gal})} = \int_\Gamma \alpha f_i f_j dS, \tag{20}$$

$$B_{i,j}^{(\mathrm{col})} = \beta(\mathbf{x}_i) f_j(\mathbf{x}_i), \quad B_{i,k}^{(\mathrm{gal})} = \int_\Gamma \beta f_i f_j dS. \tag{21}$$

The expressions on the left are for the collocation method, and those on the right are for the Galerkin method. In both methods, U and V are dense, and D, A, and B are highly sparse.

Let $\mathbf{x}_{j,n}$ be the nth vertex of the jth triangle, T_j. The potential and normal derivative on these triangles are each written as a linear combination of $3N$ basis functions:

$$\phi(\mathbf{x}) = \sum_{j=1}^{N}\sum_{n=1}^{3} \phi_{j,n} N_{j,n}(\mathbf{x}), \tag{22}$$

$$q(\mathbf{x}) = \sum_{j=1}^{N}\sum_{n=1}^{3} q_{j,n} N_{j,n}(\mathbf{x}), \tag{23}$$

where $\phi_{j,n}$ and $q_{j,n}$ are the potential and normal derivative at $\mathbf{x}_{j,n}$, and $N_{j,n}(\mathbf{x})$ is the corresponding linear nodal basis function. This set of basis functions allows for piecewise constant and linear potentials and normal derivatives on the triangles. The coefficients of these basis functions are organized into column vectors:

$$\boldsymbol{\phi}_{123} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \cdots & \phi_{N,1} & \phi_{N,2} & \phi_{N,3} \end{bmatrix}^{\mathrm{T}}, \tag{24}$$

$$\mathbf{q}_{123} = \begin{bmatrix} q_{1,1} & q_{1,2} & q_{1,3} & \cdots & q_{N,1} & q_{N,2} & q_{N,3} \end{bmatrix}^{\mathrm{T}}. \tag{25}$$

III. Correction Factor Matrix Method

A. Basic Approach

The linear systems given in Sec. II are solved using an iterative solver, such as GMRES [11], or a preconditioned variant. The matrix-vector product that is computed during each iteration (and accelerated using the FMM) is given by

$$\mathbf{S}\mathbf{u} = \begin{bmatrix} \mathrm{U}+\mathrm{D} & \mathrm{V} \\ \mathrm{A} & \mathrm{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}_{123} \\ \mathbf{q}_{123} \end{bmatrix}. \tag{26}$$

Consider

$$\mathbf{y} = \mathrm{A}\mathbf{x}, \tag{27}$$

where A is either U or V, and $\mathbf{x}$ is the corresponding vector of coefficients, either $\boldsymbol{\phi}_{123}$ or $\mathbf{q}_{123}$. Suppose we have another matrix, $\widetilde{\mathrm{A}}$, that approximates A:

$$\mathbf{y} = \widetilde{\mathrm{A}}\mathbf{x} + \left(\mathrm{A} - \widetilde{\mathrm{A}}\right)\mathbf{x} = \widetilde{\mathrm{A}}\mathbf{x} + \mathrm{C}\mathbf{x}. \tag{28}$$

The exact structure of $\widetilde{\mathrm{A}}$ and how to compute the matrix-vector product, $\widetilde{\mathrm{A}}\mathbf{x}$, for the collocation and Galerkin methods in $\mathrm{O}(N)$ using the FMM are given in Secs. III-B and III-C, respectively. The matrix-vector product, $\mathrm{C}\mathbf{x}$, corrects the errors from using $\widetilde{\mathrm{A}}\mathbf{x}$. The matrix, C, is called the correction factor matrix. In general, $\widetilde{\mathrm{A}}$ approximates A very well, so almost all entries in C are nearly zero. In fact, only $\mathrm{O}(N)$ entries in C need to be computed to achieve the desired accuracy, and the rest of the entries can be set to zero. Letting $\widetilde{\mathrm{C}}$ be the partially computed correction factor matrix, we have

$$\mathbf{y} \approx \widetilde{\mathbf{y}} = \widetilde{\mathrm{A}}\mathbf{x} + \widetilde{\mathrm{C}}\mathbf{x}. \tag{29}$$

Which entries to compute depends largely on the geometry of the problem. In the collocation method, only the entries corresponding to triangle/matching point pairs that are close to each other are computed, while all the others are set to zero. A collocation point, $\mathbf{y}$, is close to a triangle when $|\mathbf{y} - \mathbf{x}| < Cr$, where $\mathbf{x}$ and r are the center and radius of the triangle, and C is the close ratio. In the Galerkin method, only the entries corresponding to triangle/triangle pairs that are close to each other are computed, while all the others are set to zero. Two triangles are close when $|\mathbf{x}_2 - \mathbf{x}_1| < C\left((r_1 + r_2)/2\right)$, where $\mathbf{x}_1$, $\mathbf{x}_2$, r_1, and r_2 are the centers and radii of the two triangles (see Fig. 2). A larger close ratio leads to more nonzero entries in the correction factor matrix, while a smaller one leads to fewer. A denser correction factor matrix leads to more accurate results, so making the close ratio larger increases the accuracy of the method, but at the expense of speed. In our numerical experiments, we establish a method to get a consistent approximation whose error is lower than the errors in the other parts of the FMM/GPU-accelerated BEM.

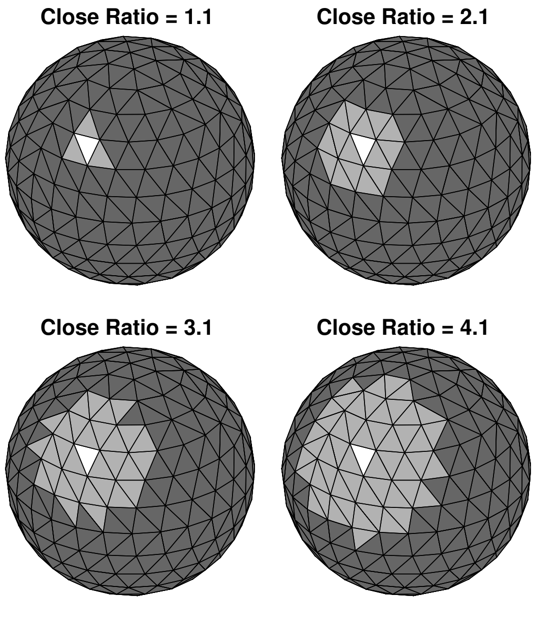

Fig. 2. In the Galerkin method, two triangles are close (and correction factors are computed) when $|\mathbf{x}_2 - \mathbf{x}_1| < C\left((r_1 + r_2)/2\right)$, where $\mathbf{x}_1$, $\mathbf{x}_2$, r_1, and r_2 are the centers and radii of the two triangles, and C is the close ratio. The light gray triangles are close to the white triangles. Four different close ratios are shown.

B. Collocation Method/Evaluation of Solution

Consider M evaluation points, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$. We want to compute the single- or double-layer potential at these points due to a potential or normal derivative on the N triangles. These M evaluation points could be the N matching points in the collocation method, and we are computing one of the matrix-vector products, $\mathrm{U}\boldsymbol{\phi}$ or $\mathrm{V}\mathbf{q}$, required by that method. Alternatively, we have already solved the problem, and we want to evaluate the solution at M different evaluation points.

Consider the following matrix-vector product:

$$\Phi = \mathrm{A}\mathbf{g}_{123}, \tag{30}$$

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}_{1,1} & \ldots & \mathrm{A}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{M,1} & \ldots & \mathrm{A}_{M,N} \end{bmatrix}, \tag{31}$$

$$\mathrm{A}_{i,j} = \begin{bmatrix} A_{i,j,1} & A_{i,j,2} & A_{i,j,3} \end{bmatrix}, \tag{32}$$

and $\mathbf{g}_{123}$ is one of the vectors of coefficients, $\boldsymbol{\phi}_{123}$ or $\mathbf{q}_{123}$. The entries in A are given by

$$A_{i,j,n} = \int_{\mathbf{x}' \in \mathrm{T}_j} N_{j,n}(\mathbf{x}') F(\mathbf{x}_i - \mathbf{x}') \, dS(\mathbf{x}'), \tag{33}$$

where $F(\mathbf{r})$ is the kernel being integrated, such as the Green's function or a derivative of the Green's function.

The entries in A are approximated using a quadrature:

$$A_{i,j,n} \approx \widetilde{A}_{i,j,n} = \sum_{l=1}^{Q_j} w_{j,l} N_{j,n}(\mathbf{p}_{j,l}) F(\mathbf{x}_i - \mathbf{p}_{j,l}), \tag{34}$$

where $\mathbf{p}_{j,l}$ and $w_{j,l}$ are the Q_j quadrature points and weights for T_j. The quadrature points we used are based on a two-dimensional Gauss-Legendre quadrature (the triangle is mapped to a quadrilateral [36], see Fig. 3). More compactly,

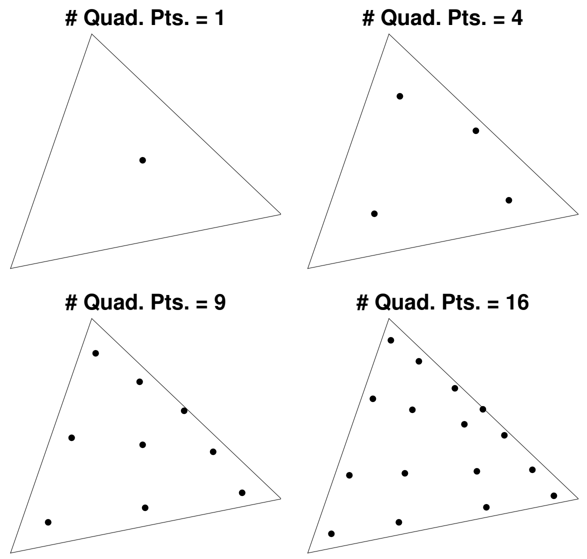

Fig. 3. The quadrature points for an example triangle. The quadrature points are based on a two-dimensional Gauss-Legendre quadrature.

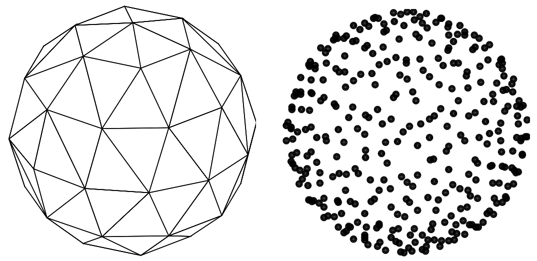

Fig. 4. On the left is a mesh of a sphere, and on the right is the set of representative point sources for the mesh.

$$\widetilde{A}_{i,j,n} = \mathrm{B}_{i,j}\mathrm{M}_{j,n}, \tag{35}$$

$$\mathrm{B}_{i,j} = \begin{bmatrix} F(\mathbf{x}_i - \mathbf{p}_{j,1}) & \ldots & F\left(\mathbf{x}_i - \mathbf{p}_{j,Q_j}\right) \end{bmatrix}, \tag{36}$$

$$\mathrm{M}_{j,n} = \begin{bmatrix} w_{j,1} N_{j,n}(\mathbf{p}_{j,1}) \\ \vdots \\ w_{j,Q_j} N_{j,n}\left(\mathbf{p}_{j,Q_j}\right) \end{bmatrix}. \tag{37}$$

Using this, we approximate the matrix-vector product, $\mathrm{A}\mathbf{g}_{123}$, using a quadrature:

$$\Phi \approx \Psi = \widetilde{\mathrm{A}}\mathbf{g}_{123}, \tag{38}$$

$$\widetilde{\mathrm{A}} = \begin{bmatrix} \widetilde{\mathrm{A}}_{1,1} & \ldots & \widetilde{\mathrm{A}}_{1,N} \\ \vdots & \ddots & \vdots \\ \widetilde{\mathrm{A}}_{M,1} & \ldots & \widetilde{\mathrm{A}}_{M,N} \end{bmatrix}, \tag{39}$$

$$\widetilde{\mathrm{A}}_{i,j} = \begin{bmatrix} \widetilde{A}_{i,j,1} & \widetilde{A}_{i,j,2} & \widetilde{A}_{i,j,3} \end{bmatrix}. \tag{40}$$

Plugging in Eq. (35) and rearranging, we have

$$\Psi = \mathrm{B}\mathrm{M}\mathbf{g}_{123}, \tag{41}$$

$$\mathrm{B} = \begin{bmatrix} \mathrm{B}_{1,1} & \dots & \mathrm{B}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{B}_{M,1} & \dots & \mathrm{B}_{M,N} \end{bmatrix}, \tag{42}$$

$$\mathrm{M} = \begin{bmatrix} \mathrm{M}_{1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathrm{M}_{N} \end{bmatrix}, \tag{43}$$

$$\mathrm{M}_j = \begin{bmatrix} \mathrm{M}_{j,1} & \mathrm{M}_{j,2} & \mathrm{M}_{j,3} \end{bmatrix}. \tag{44}$$

The mapping matrix, M, maps the potential or normal derivative on the triangles to the source strengths of a set of representative point sources (see Fig. 4). This set is the union of the quadrature points for all the triangles in the mesh. The mapping matrix is a block diagonal matrix, and is highly sparse. The matrix, B, computes the potential at the evaluation points due to these point sources. This step is accelerated using the FMM, and runs in $\mathrm{O}(N)$.

C. Galerkin Method

In the Galerkin method, the matrix-vector products, $\mathrm{V}\mathbf{q}$ and $\mathrm{U}\boldsymbol{\phi}$, compute the integral over the boundary of the single- and double-layer potentials when weighted by the basis functions.

Consider the following matrix-vector product:

$$\mathbf{I} = \mathrm{A}\mathbf{g}_{123}, \tag{45}$$

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}_{1,1} & \dots & \mathrm{A}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{M,1} & \dots & \mathrm{A}_{M,N} \end{bmatrix}, \tag{46}$$

$$\mathrm{A}_{i,j} = \begin{bmatrix} A_{i,j,1,1} & A_{i,j,1,2} & A_{i,j,1,3} \\ A_{i,j,2,1} & A_{i,j,2,2} & A_{i,j,2,3} \\ A_{i,j,3,1} & A_{i,j,3,2} & A_{i,j,3,3} \end{bmatrix}. \tag{47}$$

The entries in A are given by

$$\begin{aligned} A_{i,j,m,n} &= \int_{\mathbf{x}\in\mathrm{T}_i} N_{i,m}(\mathbf{x}) \int_{\mathbf{x}'\in\mathrm{T}_j} N_{j,n}(\mathbf{x}') \\ &\times \; F(\mathbf{x}-\mathbf{x}')\, dS(\mathbf{x}')\, dS(\mathbf{x}), \end{aligned} \tag{48}$$

where $F(\mathbf{r})$ is the kernel being integrated, such as the Green's function or a derivative of the Green's function. The inside integral computes the potential due to the potential or normal derivative, $N_{j,n}(\mathbf{x}')$, on T_j. The outside integral multiplies this potential by $N_{i,m}(\mathbf{x})$, and integrates this product over T_i.

The entries in A are approximated using a quadrature:

$$\begin{aligned} A_{i,j,m,n} &\approx \widetilde{A}_{i,j,m,n} \\ &= \sum_{k=1}^{Q_i} w_{i,k} N_{i,m}(\mathbf{p}_{i,k}) \sum_{l=1}^{Q_j} w_{j,l} N_{j,n}(\mathbf{p}_{j,l}) \\ &\times \; F(\mathbf{p}_{i,k}-\mathbf{p}_{j,l}), \end{aligned} \tag{49}$$

where $\mathbf{p}_{j,l}$ and $w_{j,l}$ are the Q_j quadrature points and weights for T_j, and $\mathbf{p}_{i,k}$ and $w_{i,k}$ are the Q_i quadrature points and weights for T_i. More compactly,

$$\widetilde{A}_{i,j,m,n} = \mathrm{M}_{i,m}^{\mathrm{T}} \mathrm{B}_{i,j} \mathrm{M}_{j,n}, \tag{50}$$

where the matrices, $\mathrm{M}_{i,m}$ and $\mathrm{M}_{j,n}$, are given by Eq. (37), and

$$\mathrm{B}_{i,j} = \begin{bmatrix} F(\mathbf{p}_{i,1}-\mathbf{p}_{j,1}) & \dots & F(\mathbf{p}_{i,1}-\mathbf{p}_{j,Q_j}) \\ \vdots & \ddots & \vdots \\ F(\mathbf{p}_{i,Q_i}-\mathbf{p}_{j,1}) & \dots & F(\mathbf{p}_{i,Q_i}-\mathbf{p}_{j,Q_j}) \end{bmatrix}. \tag{51}$$

Using this, we approximate the matrix-vector product, $\mathrm{A}\mathbf{g}_{123}$, using a quadrature:

$$\mathbf{I} \approx \mathbf{J} = \widetilde{\mathrm{A}}\mathbf{g}_{123}, \tag{52}$$

$$\widetilde{\mathrm{A}} = \begin{bmatrix} \widetilde{\mathrm{A}}_{1,1} & \dots & \widetilde{\mathrm{A}}_{1,N} \\ \vdots & \ddots & \vdots \\ \widetilde{\mathrm{A}}_{M,1} & \dots & \widetilde{\mathrm{A}}_{M,N} \end{bmatrix}, \tag{53}$$

$$\widetilde{\mathrm{A}}_{i,j} = \begin{bmatrix} \widetilde{A}_{i,j,1,1} & \widetilde{A}_{i,j,1,2} & \widetilde{A}_{i,j,1,3} \\ \widetilde{A}_{i,j,2,1} & \widetilde{A}_{i,j,2,2} & \widetilde{A}_{i,j,2,3} \\ \widetilde{A}_{i,j,3,1} & \widetilde{A}_{i,j,3,2} & \widetilde{A}_{i,j,3,3} \end{bmatrix}. \tag{54}$$

Plugging in Eq. (50) and rearranging, we have

$$\mathbf{J} = \mathrm{M}^{\mathrm{T}}\mathrm{B}\mathrm{M}\mathbf{g}_{123}, \tag{55}$$

where the matrices, M and B, are given by Eqs. (43) and (42), respectively. In this expression, the quadrature points for all N triangles are treated a point sources and assigned source strengths (i.e., $\mathrm{M}\mathbf{g}_{123}$). Next, the potential due to these point sources are evaluated at the same set of quadrature points (i.e., $\mathrm{BM}\mathbf{g}_{123}$). This step is accelerated using the FMM, and runs in $\mathrm{O}(N)$. Finally, the potentials are weighted and summed to obtain the integrals over the triangles (i.e., $\mathrm{M}^{\mathrm{T}}\mathrm{BM}\mathbf{g}_{123}$).

D. Extension to Other Kernels

We have organized our BEM software in such a way that any kernel-specific components are separate from the rest of the BEM code. In fact, there are two such components in our software. First are the routines for computing the boundary integrals in Eqs. (17) – (21). Second are the routines for computing potentials and gradients due to monopole and dipole sources. Exactly how these are implemented is not important to the rest of the software. In effect, these pieces are treated as black boxes. In practice, we use the subdivision and scaling method in [37] to compute the boundary integrals, and the FMM to compute potentials and gradients. We use the FMM because of the linear scaling, and because FMM libraries that implement the Laplace kernel are readily available. However, other fast algorithms or hardware-based methods could be used instead. The fact that these two components are treated as black boxes also allows for other kernels to be considered. For example, our BEM software could be extended to the Helmholtz equation, which is present in acoustics and other electromagnetics problems, as soon as black-box routines for the Helmholtz kernel are developed. In this paper, though, we focus our attention on the Laplace equation only.

Vertex List

x_1	x_2	$\dots$	x_{N_vert}
y_1	y_2	$\dots$	y_{N_vert}
z_1	z_2	$\dots$	z_{N_vert}

Connectivity List

$v_{1,1}$	$v_{2,1}$	$\dots$	$v_{N_\text{tri},1}$
$v_{1,2}$	$v_{2,2}$	$\dots$	$v_{N_\text{tri},2}$
$v_{1,3}$	$v_{2,3}$	$\dots$	$v_{N_\text{tri},3}$

Triangle Pairs

i_1	i_2	$\dots$	i_{N_pair}
j_1	j_2	$\dots$	j_{N_pair}

Fig. 5. On the CPU, the pairs of triangles are stored as pairs of indices, which reference values stored in the list of vertices and the connectivity list.

Triangle Pairs

$x_{1,1,1}$	$x_{2,1,1}$	$\dots$	$x_{N_\text{pair},1,1}$
$y_{1,1,1}$	$y_{2,1,1}$	$\dots$	$y_{N_\text{pair},1,1}$
$z_{1,1,1}$	$z_{2,1,1}$	$\dots$	$z_{N_\text{pair},1,1}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$x_{1,1,3}$	$x_{2,1,3}$	$\dots$	$x_{N_\text{pair},1,3}$
$y_{1,1,3}$	$y_{2,1,3}$	$\dots$	$y_{N_\text{pair},1,3}$
$z_{1,1,3}$	$z_{2,1,3}$	$\dots$	$z_{N_\text{pair},1,3}$
$x_{1,2,1}$	$x_{2,2,1}$	$\dots$	$x_{N_\text{pair},2,1}$
$y_{1,2,1}$	$y_{2,2,1}$	$\dots$	$y_{N_\text{pair},2,1}$
$z_{1,2,1}$	$z_{2,2,1}$	$\dots$	$z_{N_\text{pair},2,1}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$x_{1,2,3}$	$x_{2,2,3}$	$\dots$	$x_{N_\text{pair},2,3}$
$y_{1,2,3}$	$y_{2,2,3}$	$\dots$	$y_{N_\text{pair},2,3}$
$z_{1,2,3}$	$z_{2,2,3}$	$\dots$	$z_{N_\text{pair},2,3}$

Fig. 6. On the GPU, the coordinates of the two triangles in each pair are stored explicitly.

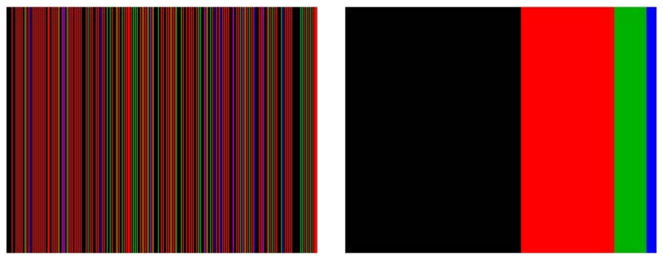
Fig. 7. The double surface integrals in the Galerkin method are sorted by case (i.e., zero-, one-, two-, and three-touch) before sending them to the GPU to avoid branch divergence. On the left is the list of integrals before sorting, and on the right is the list after sorting. Black, red, green, and blue correspond to zero-, one-, two-, and three-touch integrals, respectively.

IV. GPU Acceleration of Boundary Integrals

There are two versions of the code for computing the integrals needed by the BEM. The CPU version, written in C++ and parallelized using OpenMP, ran on an Intel Xeon E5-2698 v3 (32 cores). This version was originally developed in [37]. The GPU version, written in CUDA, ran on an NVIDIA Tesla K40. Because C++ and CUDA are largely the same programming language, the two versions are almost the same. However, because of the unique nature of GPU programming, several changes had to be made so that the CUDA code could take full advantage of the GPU hardware.

In the collocation method, each integral corresponds to a triangle/collocation point pair, while in the Galerkin method, each integral corresponds to a triangle/triangle pair. In the CPU version, there were 32 threads, each running independently from the others on one of the 32 available cores. The N_pair pairs were divided up into 32 equally sized chunks. Each thread received one chunk, and computed all the integrals corresponding to the pairs in that chunk. In the GPU version, there was one thread per pair, meaning there were N_pair threads. Each thread was responsible for computing one integral. The block size was 256, so there were $N_\text{pair}/256$ blocks.

In the CPU version, the data structures are compact (see Fig. 5). The mesh is stored as a list of vertices and a connectivity list. In the collocation method, there is also a list of collocation points. The CPU code accepts a list of index pairs. In the collocation method, one index in the pair corresponds to a triangle, and the other index in the pair corresponds to a collocation point. In the Galerkin method, the two indices in the pair correspond to two triangles. Because global memory access on the CPU is fast due to the many layers of cache, this combination of data structures works well. However, global memory access on the GPU is much slower. To overcome this problem, instead of passing in a list of index pairs, the data structures are unraveled to avoid the use of indices, and a matrix of coordinates is sent to the GPU (see Fig. 6). Each column of the matrix corresponds to a pair, and each row is a coordinate of the vertices of the triangles and/or collocation point in that pair.

In the Galerkin method, the exact method for computing an integral over a pair of triangles depends on their relative geometry [37]. There are four cases: (a) the two triangles do not touch; (b) they share a vertex; (c) they share an edge; or (d) they are the same. These are called the zero-, one-, two-, and three-touch cases, respectively. In this naming scheme, the number represents how many vertices the two triangles share. The CPU version is not affected by the order of the index pairs with respect to these cases. That is, the four cases can be distributed among the index pairs in any way. This is because the 32 threads in the CPU version are working independently on separate cores. Which instructions are being executed in one thread have no effect on those in another. However, in the GPU version, this is not case. In fact, all the threads in a particular block must execute the same instructions at the same time. When a branch does occur because two or more threads in a block correspond to different cases, each thread must pause for the other to complete their work before doing their own, and vice versa. This thread divergence can lead to slowdowns on the GPU, eliminating any advantage the GPU offers. To overcome this problem, before passing any data to the GPU, the list of index pairs is sorted by the case. That is, the zero-touch cases are at the beginning of the list, followed by the one-, two-, and three-touch cases, in that order (see Fig. 7). In this manner, only three blocks (out of hundreds of thousands) suffer from thread divergence.

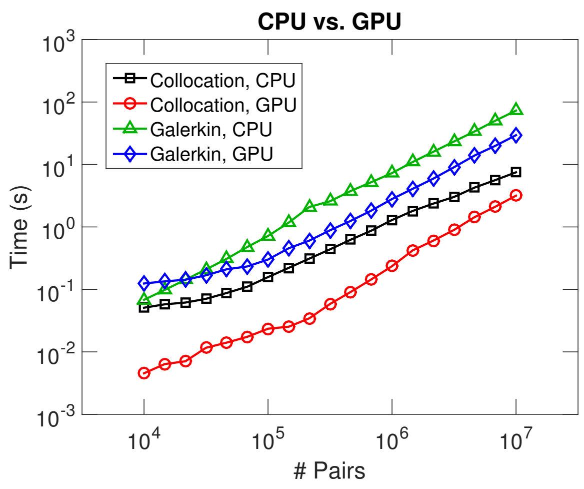

Fig. 8. The GPU computes the integrals needed by the collocation method around 2.4× faster than the CPU for large numbers of triangle/collocation point pairs. Likewise, the GPU computes the integrals needed by the Galerkin method around 2.5× faster than the CPU for large numbers of triangle/triangle pairs. The CPU was an Intel Xeon E5-2698 v3 (32 cores), and the GPU was an NVIDIA Tesla K40.

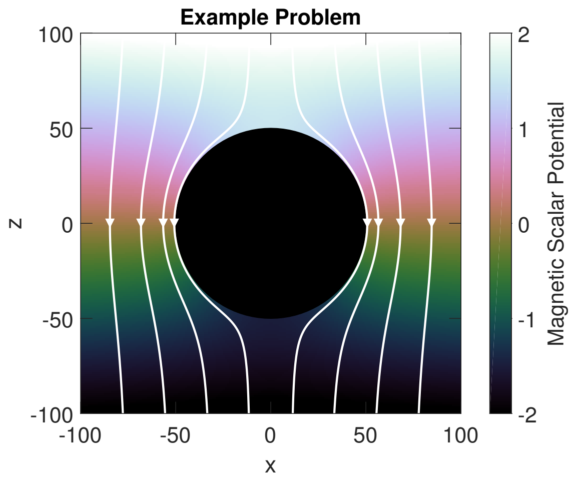

Fig. 9. The example problem is a perfectly conducting sphere placed in a uniform magnetic field. The false color shows the magnetic scalar potential, and the eight lines are field lines, which show the negative gradient of this potential (i.e., the magnetic field). The sphere "shunts" the magnetic field out, causing the magnetic field to be completely tangential at the surface.

Times were measured for each version, and then plotted against N_{pair} (see Fig. 8). The GPU version was faster than the CPU version for all values of N_{pair}. The GPU computed the integrals needed by the collocation method around 2.4× faster than the CPU for large N_{pair}. Likewise, the GPU computed those needed by the Galerkin method around 2.5× faster. The speedup for the Galerkin integrals is slightly higher because computing these integrals is more computationally intensive, and the I/O overhead was not as much of an issue. In both cases, the GPU versions provided large speedups, especially considering that the CPU had 32 cores.

V. Numerical Example: Perfectly Conducting Sphere in Uniform Magnetic Field

A. Problem Description

We ran a series of computational experiments to verify the accuracy and characterize the performance of the correction factor matrix method described in Sec. III. The example problem we used is shown in Fig. 9. A perfectly conducting sphere of radius, a, is centered at the origin, and a uniform magnetic field, $\mathbf{B}^{(\text{imp})} = B_0\mathbf{a}_z$, is imposed. We set $a = 50$ and $B_0 = -1/50$. On the surface of a perfect conductor, the normal component of the magnetic field is zero. The sphere, therefore, distorts the imposed magnetic field by creating an induced magnetic field, $\mathbf{B}^{(\text{ind})}$, which cancels out the normal component of the imposed magnetic field on the surface of the sphere. The problem, then, is to compute $\mathbf{B}^{(\text{ind})}$. Since there are no free currents in the volume around the sphere, the induced magnetic field is curl-free, and so can be expressed as a gradient field:

$$\mathbf{B}^{(\text{ind})} = -\nabla\phi_{\text{M}}^{(\text{ind})}. \tag{56}$$

Likewise, because any magnetic field is divergence-free, the induced magnetic scalar potential, $\phi_{\text{M}}^{(\text{ind})}$, satisfies

$$\nabla^2\phi_{\text{M}}^{(\text{ind})} = 0. \tag{57}$$

We want to compute $\phi_{\text{M}}^{(\text{ind})}$ such that the total magnetic field, $\mathbf{B} = \mathbf{B}^{(\text{imp})} + \mathbf{B}^{(\text{ind})}$, has zero normal component on the boundary. In other words,

$$\frac{\partial\nabla\phi_{\text{M}}^{(\text{ind})}}{\partial\mathbf{n}} = \mathbf{n}\cdot\mathbf{B}^{(\text{imp})}. \tag{58}$$

The solution to this BVP is given by

$$\phi_{\text{M}}^{(\text{ind})} = -B_0\frac{a^3}{2r^2}\cos(\theta). \tag{59}$$

To verify the accuracy of the correction factor matrix method, we can solve the BVP using the BEM. We can compute the potential at a set of validation points using the solution returned by the BEM, and then compare that to the analytical solution in Eq. (59) at the same set of points. To characterize the performance, we can time the different components of the BEM, and observe how they scale as a function of the different available solver options.

B. Accuracy

Our software has many components, each of which adds some error to the solution. Our experiments, therefore, sought to explore these sources of error, and to determine how best to set the available parameters to achieve the desired accuracy while maintaining good performance. There are several sources of error, namely:

1) The modeling error from using real-world measurements to construct the models.
2) The geometric error from discretizing the boundary using planar triangular elements.
3) The error from approximating the potential and normal derivative on the boundary using piecewise constant and linear basis functions.
4) The accuracy of the boundary integrals.
5) The error from using the correction factor matrix method, which is controlled by:

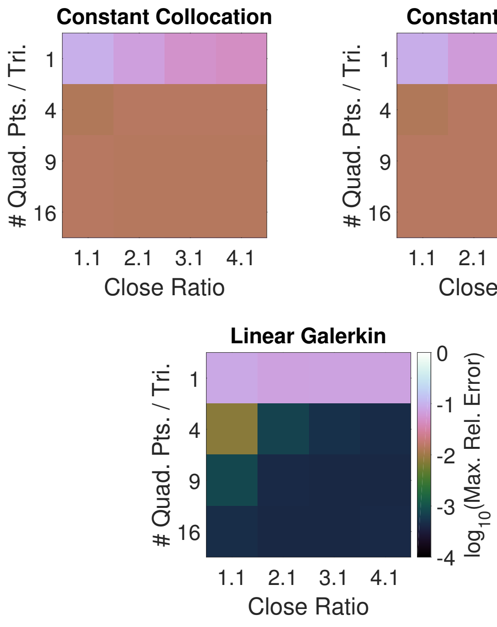

Fig. 10. The maximum relative error of the example problem as a function of the number of quadrature points per triangle and the close ratio.

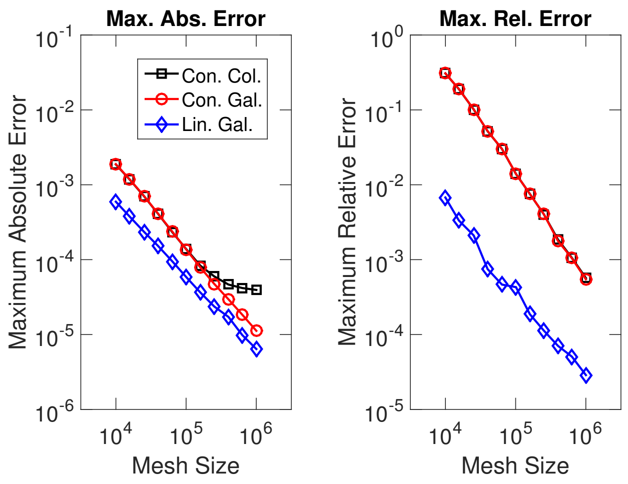

Fig. 11. The maximum absolute (left) and relative (right) errors for the example problem as a function of mesh size for the three solution methods. Constant collocation and constant Galerkin have similar error curves, while linear Galerkin gives the lowest errors.

a) the number of quadrature points per triangle; and
b) the close ratio, which determines the density of the correction factor matrix.

6) The error from the multipole and local expansions and translations in the FMM, which is controlled by the truncation number and the translation schemes used.
7) The error from the iterative solver, which is controlled by which solver is used (e.g., GMRES) and the chosen tolerance.

The modeling error does not apply to the example problem. There is geometric error because triangular elements are unable to model the smooth surface of the sphere. The approximation error is governed by the type of basis functions used (i.e., piecewise constant and linear basis functions) and the mesh size (i.e., how many basis functions are used). We explored the behavior of the approximation error by comparing the errors from using either piecewise constant or piecewise linear basis functions, and also by varying the mesh size. Figs. 2 and 4 show the meshes we used. The mesh size ranged from 10,000 triangles to over 1,000,000. The entries in the correction factor matrix should be computed accurately. For the collocation method, the required single surface integrals can be computed exactly, so there is no error. For the Galerkin method, we use the subdivision and scaling method in [37] to compute the required double surface integrals. The error from the FMM was controlled by selecting an appropriate truncation number, p. In our experiments, we chose $p = 20$. We used the unpreconditioned GMRES with a relative tolerance of 10^{-5}.

This leaves two sources of error: the number of quadrature points per triangle, Q, and the close ratio, C. These two sources of error are highly intertwined, and must be set together to achieve the desired accuracy. In general, for fewer quadrature points per triangle, larger close ratios are needed. Likewise, for more quadrature points per triangle, smaller close ratios are needed. A balance must be struck between these two parameters. To study the effect of these two parameters on the accuracy, we solved the example problem several times, holding all parameters constant, but varying the number of quadrature points per triangle and the close ratio. The mesh had 101,184 triangles, and we varied $Q = 1, 4, 9, 16$ and $C = 1.1, 2.1, 3.1, 4.1$. For each combination of Q and C, we computed the maximum relative error at a set of validation points. Fig. 10 shows this error as a function of the two parameters. As expected, increasing Q or C improves the accuracy. Also, when one parameter is decreased, the other can be increased to offset any drop in accuracy. Based on this experiment, we chose $Q = 9$ and $C = 2.1$ as the best combination in terms of accuracy and performance. This yielded 1% error for constant collocation and constant Galerkin, and 0.1% error for linear Galerkin. Using these values, we computed the maximum absolute and relative errors as a function of mesh size, N (see Fig. 11). As expected, all errors decrease as N increases. Constant collocation and constant Galerkin yield similar errors, dropping below 0.1% for the largest mesh sizes, while linear Galerkin yields much lower errors, dropping below 0.01%.

C. Performance

The code is divided into four pieces: (1) determining which entries in the correction factor matrix to compute, and then computing them; (2) computing the right-hand side; (3) solving the system; and (4) evaluating the solution at the evaluation points. To characterize the performance of each piece, we solved the example problem for different mesh sizes for the three solution methods (constant collocation, constant Galerkin, and linear Galerkin). Times were measured for each run, and then plotted as a function of mesh size (see Fig. 12).

The runs were done on a single GPU node on the Army Research Laboratory (ARL) Excalibur supercomputer. The node had an Intel Xeon E5-2698 v3 (16 cores $\times$ 2 threads/core $=$ 32 threads), 256 GB of RAM, and an NVIDIA Tesla K40.

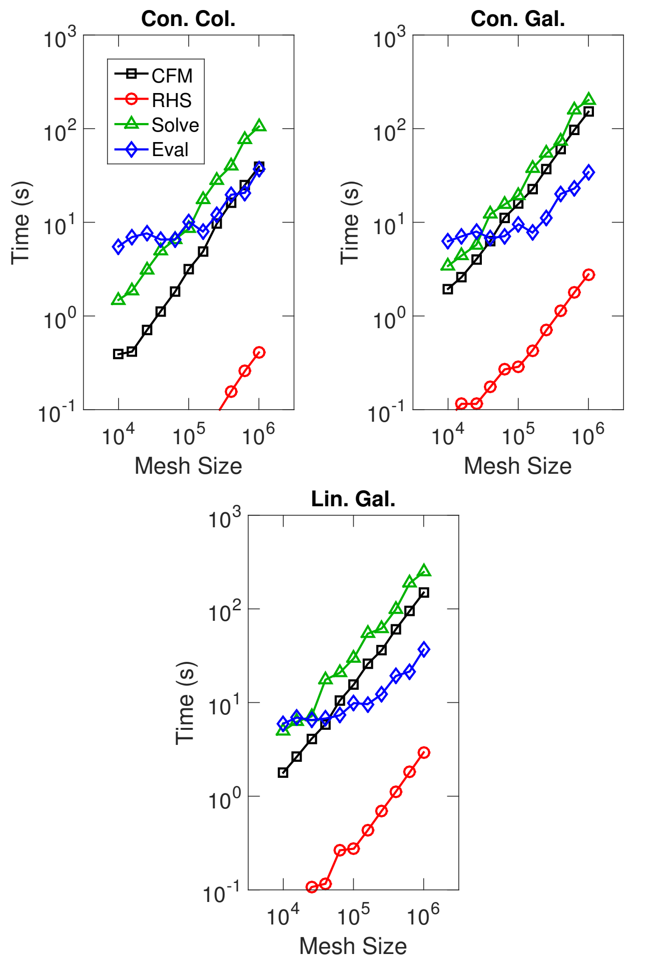

Fig. 12. The performance of the four components of our code (correction factor matrix, right-hand side, solve, and evaluation) for the example problem as a function of mesh size for the three solution methods.

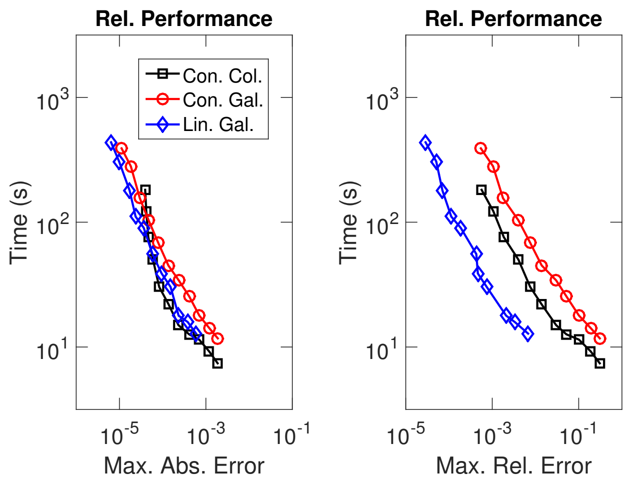

Fig. 13. The total time (sum of the times for all four components of our code) as a function of the maximum absolute (left) and relative (right) errors for the three solution methods. Linear Galerkin provides the best accuracy in the least amount of time.

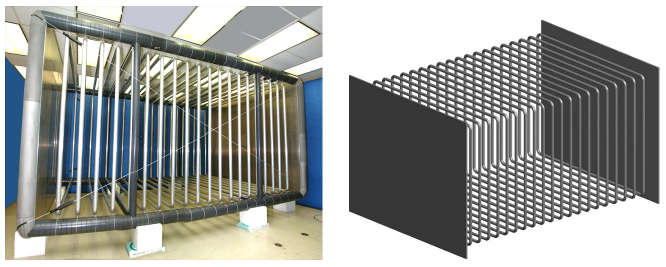

Fig. 14. The electric-field cage as constructed (left) and as modeled (right).

The code was written in a combination of Fortran, C, and C++. MATLAB was used as a glue to connect the different pieces of code together, but otherwise did very little work. The most computationally intensive code was parallelized using OpenMP to take advantage of all of the available cores on the machine. The computation of the boundary integrals and correction factors was accelerated on the GPU (see Sec. IV). The FMM was also accelerated by computing and summing the local interactions on the GPU. The GPU code was written in CUDA.

Based on the analysis from Sec. V-B, we used $Q = 9$ and $C = 2.1$. These parameters could be decreased (e.g., $Q = 4$ and $C = 1.1$) to achieve better performance at the expense of accuracy. For example, this could be done during the early stages of engineering design, where new ideas need to be tested rapidly, and answers do not need to be perfect. On the other hand, they could be increased (e.g., $Q = 16$ and $C = 3.1$) when higher accuracy is required, e.g., during the end stages of design.

Timing scaled as $\mathrm{O}(N)$ or $\mathrm{O}(N \log(N))$ for all four pieces of the code. In particular, for $N = 1,002,248$, the solve step of constant collocation ran in 2.6 minutes, constant Galerkin ran in 5.0 minutes, and linear Galerkin ran in 6.1 minutes. The difference in performance between the three solution methods is largely due to the computation of the correction factors. This makes sense: computing the double surface integrals in the two Galerkin methods is much more computationally expensive than computing the single surface integrals in the collocation method.

For the same mesh size, constant Galerkin takes longer than constant collocation, and linear Galerkin takes longer than both of these. However, linear Galerkin is much more accurate than the other two. To determine which method is fastest for the same level of accuracy and should be used, we plotted the time (sum of the times for all four components of our code) as a function of the maximum absolute and relative errors for the three solution methods (see Fig. 13). Linear Galerkin provides the best accuracy in the least amount of time. This is because linear Galerkin provides lower errors at smaller mesh sizes, and these smaller mesh sizes take shorter amounts of time to solve.

VI. Numerical Example: Electric-Field Cage

A. Problem Description

The ARL electric-field cage was constructed in 2005 as a calibration and characterization chamber for electric-field sensors [38]. It has been used to characterize many ARL and third-party sensors [39]. The cage, shown in Fig. 14 on the left, is effectively a large parallel plate capacitor with guard tubes

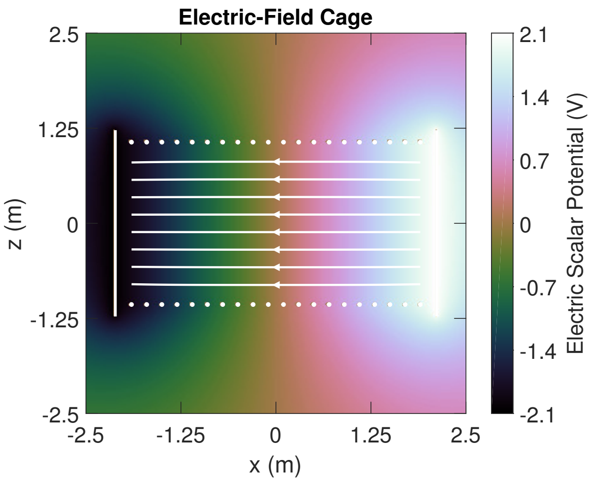

Fig. 15. The false color shows the electric scalar potential in and around the electric-field cage, and the eight lines are electric-field lines that run between the two end plates. The guard tubes control the fringing fields, producing an electric field inside the cage that is nearly uniform.

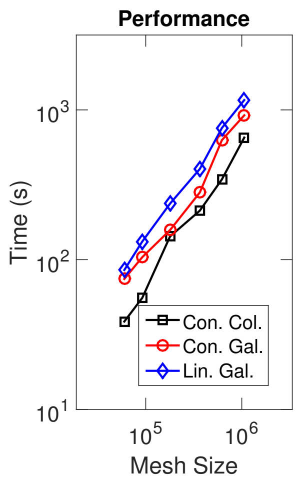

Fig. 16. The total time (sum of the times for all four components of our code) as a function of mesh size for the electric-field cage example problem using the three solution methods.

to control the fringing fields. The aluminum end plates are 1/8 in thick, 10 ft wide, and 8 ft high. They are separated by 4.2 m. The 20 aluminum guard tubes are spaced evenly between the end plates. Each one has a diameter of 2 in, is 9 ft wide and 7 ft high, and has an elbow radius of 3 in. A model of the cage can be seen in Fig. 14 on the right. The model differs from reality in two ways: the two end plates were modeled as 1 in thick instead of 1/8 in thick; and the non-conducting structural components of the cage and the surrounding lab were omitted. For this example problem, the $+x$ end plate was energized to 2.1 V and the $-x$ end plate was energized to -2.1 V (i.e., bipolar mode). The guard tubes are energized to voltages that enforce a linear gradient field between the end plates, resulting in a nearly uniform electric field near the center of the cage. For the particular voltages chosen, the strength of this field was 1 V/m (see Fig. 15).

B. Performance

The geometry of the electric-field cage is much more complicated than in the previous example problem. In particular, the cage has multiple disconnected boundaries, and some of these have holes. The BEM was used to solve for the charge distribution on the surface of these boundaries, which was then used to compute the electric field in the volume around the cage. The parameters, Q and C, were chosen to be 4 and 2.1, respectively. The truncation number in the FMM was set to $p = 12$, and the relative tolerance of the iterative solver was set to 10^{-4}. The other solver options were the same as in the previous example problem. The cage was solved using several levels of discretization. The mesh size ranged from 59,824 to 1,061,128. Fig. 16 shows the total time of the solve and evaluation steps as a function of mesh size for the three solution methods. As seen in this graph, the computation time scales nearly linearly with the mesh size. In particular, for the $N = 1,050,680$ case, the constant collocation, constant Galerkin, and linear Galerkin methods took 10.9, 15.3, and 19.4 minutes, respectively.

VII. Conclusion

An FMM/GPU-accelerated BEM for computational magnetics and electrostatics problems via the Laplace equation was presented. The unaccelerated BEM suffers from a few problems. The system matrix is dense, requiring $\mathrm{O}\left(N^2\right)$ storage, where N is the number of boundary elements, and solving the system by direct means (e.g., LU decomposition) requires $\mathrm{O}\left(N^3\right)$ operations. These quadratic and cubic costs can effectively restrict the size of a problem that can be tackled on a given machine in a reasonable amount of time. The FMM can be used to reduce these scalings to linear, and allows for very large problem sizes. However, the FMM is usually designed around monopole and dipole sources, not the integral expressions that appear in the BEM. Moreover, the FMM for monopole and dipole sources has been studied extensively by many previous authors, and implementations are widely available and highly optimized. Therefore, instead of modifying the FMM to account for these expressions, we have developed a method that allows for the FMM to be used unchanged to accelerate the BEM. The correction factor matrix method works by mapping the potential and normal derivative on the boundary to a set of representative monopole and dipole sources, and plugging these into the FMM, which is effectively treated as a black box. Any inaccuracies from this step are corrected by computing and adding correction factors. This paper stated the problem, derived and described the correction factor matrix method, and studied the accuracy and performance of the method in the context of two example problems. In particular, by properly setting the available parameters, errors were easily reduced to below 0.01%. The method was also shown to scale linearly and be able to solve problems with a million elements in only minutes on a single workstation.

VIII. Acknowledgements

Ross Adelman was supported under cooperative agreement W911NF1420118 between the Army Research Laboratory and the University of Maryland, College Park, with David Hull and

Stephen Vinci as technical monitors. Nail Gumerov and Ramani Duraiswami were partially supported by the same cooperative agreement, and partially by NSF award CMMI1250187.

References

[1] R. F. Harrington, *Field Computation by Moment Methods*. New York: The Macmillan Company, 1968.

[2] C. Pozrikids, *A Practical Guide to Boundary Element Methods with the Software Library BEMLIB*. New York: Chapman & Hall/CRC, 2002.

[3] W. C. Gibson, *The Method of Moments in Electromagnetics*. New York: Chapman & Hall/CRC, 2008.

[4] E. Lezar and D. B. Davidson, "GPU-accelerated method of moments by example: monostatic scattering," *IEEE Antennas and Propagation Magazine*, vol. 52, no. 6, pp. 120 – 135, December 2010.

[5] Z. Zhang, I. D. Mayergoyz, N. A. Gumerov, and R. Duraiswami, "Numerical analysis of plasmon resonances in nanoparticles based on fast multipole method," *IEEE Transactions on Magnetics*, vol. 43, no. 4, pp. 1465 – 1468, April 2007.

[6] R. A. Kerr, "A quickie birth for Jupiters and Saturns," *Science*, vol. 298, no. 5599, pp. 1698 – 1699, November 2002.

[7] N. A. Gumerov and R. Duraiswami, "Efficient FMM accelerated vortex methods in three dimensions via the Lamb-Helmholtz decomposition," *Journal of Computational Physics*, vol. 240, pp. 310 – 328, May 2013.

[8] C. C. Chabalko and B. Balachandran, "Implementation and benchmarking of two-dimensional vortex interactions on a graphics processing unit," *Journal of Aerospace Information Systems*, vol. 11, no. 6, pp. 372 – 385, 2014.

[9] K. Nabors and J. White, "FastCap: a multipole accelerated 3-d capacitance extraction program," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 11, pp. 1447 – 1459, 1991.

[10] M. Kamon, M. J. Tsuk, and J. K. White, "FASTHENRY: a multipole-accelerated 3-D inductance extraction program," *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 9, pp. 1750 – 1758, September 1994.

[11] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, 1986.

[12] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325 – 348, December 1987.

[13] H. Cheng, L. Greengard, and V. Rokhlin, "A fast adaptive multipole algorithm in three dimensions," *Journal of Computational Physics*, vol. 155, no. 2, pp. 468 – 498, November 1999.

[14] N. A. Gumerov and R. Duraiswami, "Fast multipole methods on graphics processors," *Journal of Computational Physics*, vol. 227, no. 18, pp. 8290 – 8313, September 2008.

[15] Q. Hu, N. A. Gumerov, and R. Duraiswami, "Scalable fast multipole methods on distributed heterogeneous architectures," *In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, 2011.

[16] R. Yokota, L. Barba, T. Narumi, and K. Yasuoka, "Petascale turbulence simulation using a highly parallel fast multipole method on GPUs," *Computer Physics Communications*, vol. 184, no. 3, pp. 445 – 455, 2013.

[17] http://www.cims.nyu.edu/cmcl/software.html.

[18] G. Of, O. Steinbach, and W. L. Wendland, "Applications of a fast multipole Galerkin in boundary element method in linear elastostatics," *Computing and Visualization in Science*, vol. 8, no. 3 - 4, pp. 201 – 209, December 2005.

[19] A. D. Pham, S. Mouhoubi, M. Bonnet, and C. Chazallon, "Fast multipole method applied to symmetric Galerkin boundary element method for 3D elasticity and fracture problems," *Engineering Analysis with Boundary Elements*, vol. 36, no. 12, pp. 1838 – 1847, December 2012.

[20] N. A. Gumerov and R. Duraiswami, "A broadband fast multipole accelerated boundary element method for the 3D Helmholtz equation," *J. Acoust. Soc. Am.*, vol. 125, no. 1, pp. 191 – 205, January 2009.

[21] N. A. Gumerov, A. E. O'Donovan, R. Duraiswami, and D. N. Zotkin, "Computation of the head-related transfer function via the fast multipole accelerated boundary element method and its spherical harmonic representation," *J. Acoust. Soc. Am.*, vol. 127, pp. 370 – 386, January 2010.

[22] B. Michiels, J. Fostier, I. Bogaert, and D. D. Zutter, "Full-wave simulations of electromagnetic scattering problems with billions of unknowns," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 2, pp. 796 – 799, February 2015.

[23] K. C. Donepudi, J. Song, J.-M. Jin, G. Kang, and W. C. Chew, "A novel implementation of multilevel fast multipole algorithm for higher order Galerkin's method," *IEEE Transactions on Antennas and Propagation*, vol. 48, no. 8, pp. 1192 – 1197, August 2000.

[24] D. Dault and B. Shanker, "A mixed potential MLFMA for higher order moment methods with application to the generalized method of moments," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 2, pp. 650 – 662, February 2016.

[25] J. D. Jackson, *Classical Electrodynamics*. John Wiley & Sons, Inc., 1999.

[26] A. J. Burton and G. F. Miller, "The application of integral equation methods to the numerical solution of some exterior boundary-value problems," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 323, no. 1553, pp. 201 – 210, 1971.

[27] S. M. Rao, A. W. Glisson, D. Wilton, and B. Vidula, "A simple numerical solution procedure for statics problems involving arbitrary-shaped surfaces," *IEEE Transactions on Antennas and Propagation*, vol. 27, no. 5, pp. 604 – 608, September 1979.

[28] S. M. Rao, D. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 3, pp. 409 – 418, May 1982.

[29] D. R. Wilton, S. M. Rao, A. W. Glisson, D. H. Schaubert, O. Al-Bundak, and C. M. Butler, "Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains," *IEEE Transactions on Antennas and Propagation*, vol. 32, no. 3, pp. 276 – 281, March 1984.

[30] K. Davey and S. Hinduja, "Analytical integration of linear three-dimensional triangular elements in BEM," *Applied Mathematical Modelling*, vol. 13, no. 8, pp. 450 – 461, August 1989.

[31] R. D. Graglia, "On the numerical integration of the linear shape functions times the 3-D Green's function or its gradient on a plane triangle," *IEEE Transactions on Antennas and Propagation*, vol. 41, no. 10, pp. 1448 – 1455, October 1993.

[32] J. Katz and A. Plotkin, *Low-Speed Aerodynamics*. New York, NY: Cambridge University Press, 2001.

[33] L. Li and T. F. Eibert, "Radial-angular singularity cancellation transformations derived by variable separation," *IEEE Transactions on Antennas and Propagation*, vol. 64(1), pp. 189 – 200, January 2016.

[34] R. Natarajan, "An iterative scheme for dense, complex-symmetric, linear systems in acoustics boundary-element computations," *SIAM J. Sci. Comput.*, vol. 19, no. 5, pp. 1450 – 1470, September 1998.

[35] O. O. Ademoyero, M. C. Bartholomew-Biggs, and A. J. Davies, "Computational linear algebra issues in the Galerkin boundary element method," *Computers & Mathematics with Applications*, vol. 42, no. 10 - 11, pp. 1267 – 1283, November/December 2001.

[36] S. Deng, "Quadrature Formulas in Two Dimensions," Spring 2010, University of North Carolina-Charlotte Math 5172 Lecture.

[37] R. Adelman, N. A. Gumerov, and R. Duraiswami, "Computation of Galerkin double surface integrals in the 3-D boundary element method," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 6, pp. 2389 – 2400, June 2016.

[38] D. M. Hull, S. J. Vinci, and Y. Zhang, "ARL electric-field cage modeling, design and calibration," *IEEE Conference on Electromagnetic Field Computation*, 2006.

[39] S. M. Heintzelman and D. M. Hull, "Characterization and analysis of electric-field sensors," *IEEE Industry Applications Society Annual Meeting*, October 2015.