A Treecode-Accelerated Boundary Integral Poisson-Boltzmann Solver for Electrostatics of Solvated Biomolecules

Weihua Geng[a], Robert Krasny[b,*]

[a] *Department of Mathematics, University of Alabama, Tuscaloosa, AL 35487 USA*
[b] *Department of Mathematics, University of Michigan, Ann Arbor, MI 48109 USA*

Abstract

We present a treecode-accelerated boundary integral (TABI) solver for electrostatics of solvated biomolecules described by the linear Poisson-Boltzmann equation. The method employs a well-conditioned boundary integral formulation for the electrostatic potential and its normal derivative on the molecular surface. The surface is triangulated and the integral equations are discretized by centroid collocation. The linear system is solved by GMRES iteration and the matrix-vector product is carried out by a Cartesian treecode which reduces the cost from $O(N^2)$ to $O(N \log N)$, where N is the number of faces in the triangulation. The TABI solver is applied to compute the electrostatic solvation energy in two cases, the Kirkwood sphere and a solvated protein. We present the error, CPU time, and memory usage, and compare results for the Poisson-Boltzmann and Poisson equations. We show that the treecode approximation error can be made smaller than the discretization error, and we compare two versions of the treecode, one with uniform clusters and one with non-uniform clusters adapted to the molecular surface. For the protein test case, we compare TABI results with those obtained using the grid-based APBS code, and we also present parallel TABI simulations using up to eight processors. We find that the TABI solver exhibits good serial and parallel performance combined with relatively simple implementation, efficient memory usage, and geometric adaptability.

Keywords: Electrostatics; Solvated biomolecule; Poisson-Boltzmann equation; Boundary integral equation; Treecode

1. Introduction

Electrostatic interactions between a biomolecule and its solvent environment play an important role in biochemistry [1, 2]. Computing these interactions using explicit solvent models is computationally expensive, and a number of less costly implicit solvent models have been developed [3, 4]. Here we consider a model based on the linear Poisson-Boltzmann (PB) equation which treats the solute biomolecule as a low-dielectric medium with embedded atomic charges and the solvent as a high-dielectric medium with dissolved ions [5–8]. The solute in general may be a protein or more complex system for example involving membranes [9] or nucleic acids [10]. Despite the reduced cost in comparison with explicit solvent models, there is still a need to improve the efficiency of implicit solvent PB simulations [11, 12]. In the present work we address this issue by presenting a treecode-accelerated boundary integral PB solver. We start by describing the implicit solvent PB model and related numerical methods.

*Corresponding author
Email addresses: wgeng@as.ua.edu (Weihua Geng), krasny@umich.edu (Robert Krasny)

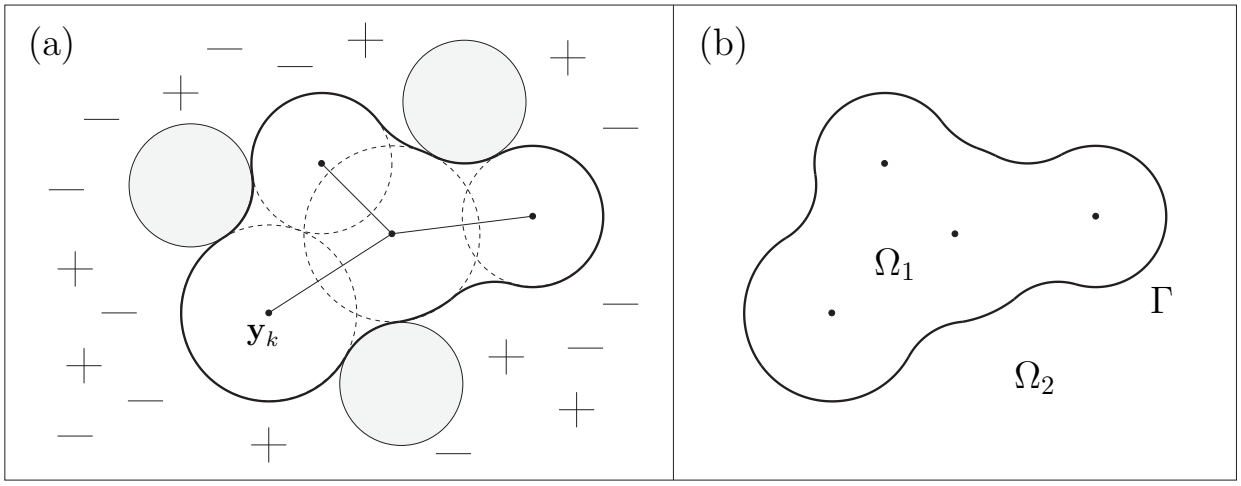

Figure 1: Solvated biomolecule (two-dimensional schematic); (a) physical model, solute atom locations $\mathbf{y}_k$, atom radii (dashed circles), solvent molecules (shaded circles), dissolved ions $(+,-)$; (b) mathematical model, interior domain Ω_1, exterior domain Ω_2, interface Γ (solvent excluded surface, molecular surface [13, 14]).

1.1. Implicit solvent Poisson-Boltzmann model

Figure 1 shows the implicit solvent model upon which this work is based. The interior domain $\Omega_1 \subset \mathbb{R}^3$ contains the solute biomolecule, and the exterior domain $\Omega_2 = \mathbb{R}^3 \setminus \overline{\Omega}_1$ contains the solvent and dissolved ions. The interface is denoted by Γ. The biomolecule is represented by a set of atomic charges Q_k at locations $\mathbf{y}_k \in \Omega_1, k = 1, \ldots, N_c$. The electrostatic potential ϕ satisfies

$$-\varepsilon_1 \nabla^2 \phi(\mathbf{x}) = \sum_{k=1}^{N_c} q_k \delta(\mathbf{x} - \mathbf{y}_k), \quad \mathbf{x} \in \Omega_1, \tag{1a}$$

$$-\varepsilon_2 \nabla^2 \phi(\mathbf{x}) + \bar{\kappa}^2 \phi(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega_2, \tag{1b}$$

where $\varepsilon_1, \varepsilon_2$ are the dielectric constants in Ω_1, Ω_2, respectively, $q_k = e_c Q_k / k_B T$ is the partial atomic charge, e_c is the electronic charge, k_B is Boltzmann's constant, T is the temperature, δ is the delta function, and κ is the Debye-Hückel parameter measuring the ionic concentration with $\kappa^2 = \bar{\kappa}^2/\varepsilon_2$. Equation (1a) is the Poisson equation with the solute charge distribution on the right side, and Equation (1b) is the linear PB equation, which reduces to the Poisson equation for $\kappa = 0\,\text{Å}^{-1}$. The interface conditions on the molecular surface are

$$\phi_1(\mathbf{x}) = \phi_2(\mathbf{x}), \quad \varepsilon_1 \frac{\partial \phi_1(\mathbf{x})}{\partial \nu} = \varepsilon_2 \frac{\partial \phi_2(\mathbf{x})}{\partial \nu}, \quad \mathbf{x} \in \Gamma, \tag{2}$$

where the subscripts $1, 2$ on ϕ denote limiting values as the interface is approached from within each domain, and ν is the outward unit normal vector on Γ. Equation (2) expresses the continuity of the potential and electric flux across the interface. The far-field boundary condition is

$$\lim_{|\mathbf{x}| \to \infty} \phi(\mathbf{x}) = 0. \tag{3}$$

In this work the interface Γ is the solvent excluded surface (also called the molecular surface) obtained by rolling a solvent sphere over the van der Waals surface of the solute [13, 14]. The goal here is to compute the potential and related quantities such as the electrostatic solvation energy used in the study of biomolecular structure [1].

1.2. Numerical methods for the Poisson-Boltzmann equation

Numerical methods for the problem described above fall into two classes, (1) grid-based methods that discretize the entire domain, e.g. [1, 15–21], and (2) boundary integral methods that discretize the molecular surface, e.g. [22–34]. We briefly discuss these approaches.

1.2.1. Grid-based methods

This class includes finite-difference and finite-element PB solvers implemented in software tools such as DelPhi [1], UHBD [15], CHARMM [16], APBS [18], and AMBER [19]. These methods require solving a sparse linear system and they employ fast iterative techniques for that purpose. Grid-based PB solvers are in widespread use, but several issues listed below constrain their performance.

1. The memory requirement for a three-dimensional grid can be prohibitively large.
2. The geometric details of the molecular surface may be obscured on a regular grid.
3. The singular atomic charges are smoothed by interpolation onto the grid.
4. The interface conditions may not be rigorously enforced on the molecular surface.
5. The far-field boundary condition is often satisfied approximately on a truncated domain.

Many techniques have been developed to address these issues including adaptive Cartesian grids [20], rigorous treatment of the interface conditions [35–38], and methods to account for the charge singularity [39, 40].

1.2.2. Boundary integral methods

Boundary integral methods alleviate some of the difficulties arising with grid-based PB solvers, as indicated below.

1. The memory requirement is reduced since the problem is formulated on the molecular surface.
2. The interface geometry can be captured more accurately using suitable boundary elements.
3. The singular atomic charges are treated analytically.
4. The interface conditions are rigorously enforced on the molecular surface.
5. The far-field boundary condition is exactly satisfied at spatial infinity.

Despite these advantages, boundary integral PB solvers encounter other difficulties such as the cost of solving a dense linear system and the need to evaluate singular integrals. These issues have been addressed using the fast multipole method [41–43], Krylov iterative techniques [44], and higher order boundary elements [45], but there is still interest in further optimizing the performance of boundary integral PB solvers.

1.2.3. The present work

We present a treecode-accelerated boundary integral (TABI) solver for the implicit solvent PB model described above. The method uses a well-conditioned boundary integral formulation for the electrostatic potential and its normal derivative on the molecular surface [22]. The surface is triangulated and the integral equations are discretized by centroid collocation with the singular term omitted [46]. The linear system is solved by GMRES iteration [25, 44] and the matrix-vector product is carried out by a Cartesian treecode [47–50] which reduces the cost from $O(N^2)$ to $O(N \log N)$, where N is the number of faces in the triangulation.

The TABI solver is applied to compute the electrostatic solvation energy in two cases, the Kirkwood sphere ($N_c = 1$ atom, with the icosahedral geodesic grid triangulation) and a solvated protein (PDB:1A63, $N_c = 2069$ atoms, with the MSMS triangulation [51]). We present the error, CPU time, and memory usage, and compare results for the PB and Poisson equations. We find

that the discretization error is $O(N^{-1/2})$ for the sphere test case and $O(N^{-1})$ for the protein test case. We show that the treecode approximation error can be made smaller than the discretization error, and we compare two versions of the treecode, one with uniform clusters and one with non-uniform clusters adapted to the molecular surface. We compare TABI results with those obtained using the grid-based APBS code [17, 18], and we also present parallel TABI simulations using up to eight processors.

To better place the TABI solver in the context of other boundary integral PB solvers, the main novelty here is that we rely on a recently developed Cartesian treecode for the screened Coulomb potential [50]. The treecode is an alternative to the fast multipole method (FMM) [41–43] for computing the matrix-vector product in the iterative solution of the discrete system. The treecode and FMM share some common features, e.g. they both use a tree data structure of particle clusters, and they employ far-field multipole expansions to approximate well-separated particle interactions. But the two methods also differ in several ways, e.g. the evaluation strategy, well-separated criterion, coordinate systems typically used, and adaptability of particle clusters. In particular, the FMM converts the multipole approximations into local approximations which are evaluated at the leaves of the tree, while the treecode can evaluate the multipole approximations at higher levels in the tree when the well-separated criterion is satisfied. The FMM has great appeal due to its $O(N)$ operation count in principle, and it has been employed in many previous boundary integral PB solvers, e.g. [23–25, 28, 31–34]. However the operation count is just one factor among several that determine the solver's CPU run time in practice. For example, memory access and communication overhead can significantly impact performance on modern serial and parallel processors, and this is where the Cartesian treecode, even with an $O(N \log N)$ operation count, may have an advantage due to its relatively simple implementation, efficient memory usage, and geometric adaptability.

We designed the TABI solver to be as simple as possible, consistent with good performance. Hence in addition to the Cartesian treecode, we employed a simple low order quadrature rule for the singular integrals, centroid collocation with the singular term omitted. This is in contrast to other PB solvers which use higher order quadrature rules and analytic expressions to evaluate the singular term; in principle those methods have a higher convergence rate, but the code tends to be more complicated and the CPU run time may be adversely affected. On the other hand, a low order quadrature rule can be made adaptive to improve performance and this is what we find here; our results for protein 1A63 converge at the rate $O(N^{-1})$, higher than the expected rate $O(N^{-1/2})$, and we attribute this to the adaptive nature of the MSMS triangulation of the molecular surface. Adaptivity also enters the TABI solver in the use of non-uniform adapted particle clusters in the treecode. Ultimately, the solver needs to parallelize well and again this is where a simpler approach can be advantageous.

The article is organized as follows. Section 2 summarizes the boundary integral formulation of the Poisson-Boltzmann implicit solvent model. In Section 3 we present the details of the treecode-accelerated boundary integral PB solver and in Section 4 we describe the two test cases. Section 5 defines the error measures used to assess the accuracy of the results. Section 6 treats the sphere test case and Section 7 treats the protein test case. A summary and conclusions are given in Section 8.

2. Boundary integral formulation

This section summarizes the boundary integral formulation of the implicit solvent PB model used in the present work [22]. Green's theorem applied to Equations (1a)-(1b) yields expressions

for the electrostatic potential in each domain,

$$\phi(\mathbf{x}) = \int_\Gamma \left[G_0(\mathbf{x},\mathbf{y}) \frac{\partial \phi(\mathbf{y})}{\partial \nu} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi(\mathbf{y}) \right] dS_\mathbf{y} + \frac{1}{\varepsilon_1} \sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad \mathbf{x} \in \Omega_1, \tag{4a}$$

$$\phi(\mathbf{x}) = \int_\Gamma \left[-G_\kappa(\mathbf{x},\mathbf{y}) \frac{\partial \phi(\mathbf{y})}{\partial \nu} + \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi(\mathbf{y}) \right] dS_\mathbf{y}, \quad \mathbf{x} \in \Omega_2, \tag{4b}$$

where $G_0(\mathbf{x},\mathbf{y})$ and $G_\kappa(\mathbf{x},\mathbf{y})$ are the Coulomb and screened Coulomb potentials,

$$G_0(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|}, \quad G_\kappa(\mathbf{x},\mathbf{y}) = \frac{e^{-\kappa|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|}. \tag{5}$$

In Equations (4a)-(4b), the normal derivative with respect to $\mathbf{y}$ is given by

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} = \nu(\mathbf{y}) \cdot \nabla_\mathbf{y} G(\mathbf{x},\mathbf{y}) = \sum_{n=1}^{3} \nu_n(\mathbf{y}) \partial_{y_n} G(\mathbf{x},\mathbf{y}), \tag{6}$$

where G represents either G_0 or G_κ. Following the steps in [22], the interface conditions yield equations for the surface potential ϕ_1 and its normal derivative $\frac{\partial \phi_1}{\partial \nu}$ on Γ,

$$\frac{1}{2}(1+\varepsilon)\,\phi_1(\mathbf{x}) = \int_\Gamma \left[K_1(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] dS_\mathbf{y} + S_1(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{7a}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right) \frac{\partial \phi_1(\mathbf{x})}{\partial \nu} = \int_\Gamma \left[K_3(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_4(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] dS_\mathbf{y} + S_2(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{7b}$$

where $\varepsilon = \varepsilon_2/\varepsilon_1$. The kernels $K_{1,2,3,4}$ are defined by

$$K_1(\mathbf{x},\mathbf{y}) = G_0(\mathbf{x},\mathbf{y}) - G_\kappa(\mathbf{x},\mathbf{y}), \quad K_2(\mathbf{x},\mathbf{y}) = \varepsilon \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}}, \tag{8a}$$

$$K_3(\mathbf{x},\mathbf{y}) = \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}} - \frac{1}{\varepsilon} \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}}, \quad K_4(\mathbf{x},\mathbf{y}) = \frac{\partial^2 G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x} \partial \nu_\mathbf{y}} - \frac{\partial^2 G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x} \partial \nu_\mathbf{y}}, \tag{8b}$$

where the normal derivative with respect to $\mathbf{x}$ is given by

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}} = -\nu(\mathbf{x}) \cdot \nabla_\mathbf{x} G(\mathbf{x},\mathbf{y}) = -\sum_{m=1}^{3} \nu_m(\mathbf{x}) \partial_{x_m} G(\mathbf{x},\mathbf{y}), \tag{9}$$

and the second normal derivative with respect to $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y} \partial \nu_\mathbf{x}} = -\sum_{m=1}^{3} \sum_{n=1}^{3} \nu_m(\mathbf{x}) \nu_n(\mathbf{y}) \partial_{x_m} \partial_{y_n} G(\mathbf{x},\mathbf{y}). \tag{10}$$

The source terms $S_{1,2}$ are defined by

$$S_1(\mathbf{x}) = \frac{1}{\varepsilon_1} \sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad S_2(\mathbf{x}) = \frac{1}{\varepsilon_1} \sum_{k=1}^{N_c} q_k \frac{\partial G_0(\mathbf{x},\mathbf{y}_k)}{\partial \nu_\mathbf{x}}. \tag{11}$$

Equations (7a)-(7b) comprise a set of coupled second kind integral equations for the surface potential ϕ_1 and its normal derivative $\frac{\partial \phi_1}{\partial \nu}$ on Γ. The electrostatic solvation energy is

$$E_{\text{sol}} = \frac{1}{2} \sum_{k=1}^{N_c} q_k \phi_{\text{reac}}(\mathbf{y}_k) = \frac{1}{2} \sum_{k=1}^{N_c} q_k \int_\Gamma \left[K_1(\mathbf{y}_k,\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{y}_k,\mathbf{y}) \phi_1(\mathbf{y}) \right] dS_\mathbf{y}, \tag{12}$$

where $\phi_{\text{reac}}(\mathbf{x}) = \phi(\mathbf{x}) - S_1(\mathbf{x})$ is the reaction potential.

3. Numerical method

In this section we present the discretization of the boundary integral equations, the treecode algorithm for the matrix-vector product, a description of how the Poisson equation ($\kappa = 0\,\text{Å}^{-1}$) is treated, and finally some coding details.

3.1. Discretization

We assume a triangulation of the molecular surface is known; examples will be discussed later. The integrals are discretized by centroid collocation [46]. Let $\mathbf{x}_i, A_i, i = 1, \ldots, N$ denote the centroids and areas of the faces in the triangulation. Then the discretized equations (7a)-(7b) have the following form for $i = 1, \ldots, N$,

$$\frac{1}{2}\left(1+\varepsilon\right)\phi_1(\mathbf{x}_i) = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_1(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu} + K_2(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right]A_j + S_1(\mathbf{x}_i), \tag{13a}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial\phi_1(\mathbf{x}_i)}{\partial\nu} = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_3(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu} + K_4(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right]A_j + S_2(\mathbf{x}_i). \tag{13b}$$

The term $j = i$ is omitted to avoid the kernel singularity; this can be motivated by recalling the definition of the principal value of a singular integral in which a neighborhood of the singularity is deleted and the limit is taken as the radius of the neighborhood tends to zero. Alternative methods for handling the singularity can be employed [22, 46], but they tend to be more complicated and we aim instead to demonstrate the capability of the present simple approach. The electrostatic solvation energy (12) is evaluated by

$$E_{sol} = \frac{1}{2}\sum_{k=1}^{N_c} q_k \sum_{j=1}^{N}\left[K_1(\mathbf{y}_k,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu} + K_2(\mathbf{y}_k,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right]A_j. \tag{14}$$

Equations (13a)-(13b) define a linear system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ contains the surface potential values $\phi_1(\mathbf{x}_i)$ and normal derivative values $\frac{\partial\phi_1}{\partial\nu}(\mathbf{x}_i)$, and $\mathbf{b}$ contains the source terms $S_1(\mathbf{x}_i), S_2(\mathbf{x}_i)$. The linear system is solved by GMRES iteration which requires a matrix-vector product in each step [44]. Since the matrix is dense, computing the product by direct summation requires $O(N^2)$ operations, which is prohibitively expensive when N is large, and in the next section we describe the treecode algorithm used to accelerate the product. Note that the source terms $S_1(\mathbf{x}_i), S_2(\mathbf{x}_i)$ in Equation (11), and the electrostatic solvation energy E_{sol} in Equation (14) amount to particle interactions between the atomic charges $\mathbf{y}_k$ and the centroids $\mathbf{x}_i$. Hence these terms require $O(N_c N)$ operations, but in the examples considered here we have $N_c << N$, so the main cost in solving the linear system arises from the matrix-vector product.

3.2. Treecode algorithm

We summarize the treecode algorithm and refer to previous work for more details [47–50]. The required sums in Equations (13a)-(13b) have the form of N-body potentials,

$$V_i = \sum_{\substack{j=1\\ j\neq i}}^{N} q_j G(\mathbf{x}_i,\mathbf{x}_j), \quad i = 1, \ldots, N, \tag{15}$$

where G is a kernel, $\mathbf{x}_i, \mathbf{x}_j$ are the centroids (also called particles in the treecode context), and q_j is a charge associated with $\mathbf{x}_j$. For example, the term involving K_1 on the right side of Equation (13a)

has the form given in Equation (15) with $q_j = \frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu} A_j$. To evaluate the potentials V_i rapidly, the particles $\mathbf{x}_i$ are divided into a hierarchy of clusters having a tree structure. The root cluster is a cube containing all the particles and subsequent levels are obtained by dividing a parent cluster into eight children [47]. The process continues until a cluster has fewer than N_0 particles (a user-specified parameter). This yields uniform clusters on each level; further below we describe a modification yielding non-uniform clusters adapted to the particle distribution.

Once the clusters are determined, the treecode evaluates the potential in Equation (15) as a sum of particle-cluster interactions,

$$V_i \approx \sum_{c \in N_i} \sum_{\mathbf{x}_j \in c} q_j G(\mathbf{x}_i, \mathbf{x}_j) + \sum_{c \in F_i} \sum_{||\mathbf{k}||=0}^{p} a^{\mathbf{k}}(\mathbf{x}_i, \mathbf{x}_c)\, m_c^{\mathbf{k}}, \tag{16}$$

where c denotes a cluster, and N_i, F_i denote the near-field and far-field of particle $\mathbf{x}_i$. The first term on the right is a direct sum for particles $\mathbf{x}_j$ near $\mathbf{x}_i$, and the second term is a pth order Cartesian Taylor approximation about the cluster center $\mathbf{x}_c$ for clusters that are well-separated from $\mathbf{x}_i$. The Taylor coefficients are given by

$$a^{\mathbf{k}}(\mathbf{x}_i, \mathbf{x}_c) = \frac{1}{\mathbf{k}!} \partial_{\mathbf{y}}^{\mathbf{k}} G(\mathbf{x}_i, \mathbf{x}_c), \tag{17}$$

and the cluster moments are given by

$$m_c^{\mathbf{k}} = \sum_{\mathbf{x}_j \in c} q_j (\mathbf{x}_j - \mathbf{x}_c)^{\mathbf{k}}. \tag{18}$$

Cartesian multi-index notation is used with $\mathbf{k} = (k_1, k_2, k_3), k_i \in \mathbb{N}, ||\mathbf{k}|| = k_1 + k_2 + k_3, \mathbf{k}! = k_1! k_2! k_3!$. A particle $\mathbf{x}_i$ and a cluster c are defined to be well-separated if the following multipole acceptance criterion (MAC) is satisfied,

$$\frac{r_c}{R} \leq \theta, \tag{19}$$

where $r_c = \max_{\mathbf{x}_j \in c} |\mathbf{x}_j - \mathbf{x}_c|$ is the cluster radius, $R = |\mathbf{x}_i - \mathbf{x}_c|$ is the particle-cluster distance, and θ is a user-specified parameter [47]. If the criterion is not satisfied, the code examines the children of the cluster recursively until the leaves of the tree are reached at which point direct summation is used. The Taylor coefficients are computed using recurrence relations [50]. In the work presented here we chose $N_0 = 500$ for the maximum size of a leaf. Results below will document the effect of the approximation order p and MAC parameter θ. This concludes the description of the treecode and next we explain some details of its application to the matrix-vector product.

3.3. Application of treecode to matrix-vector product

The matrix-vector product amounts to evaluating the sums on the right side of Equations (13a)-(13b). The kernels $K_{1,2,3,4}$ appearing there, defined in Equations (8a)-(8b), are linear combinations of the Coulomb potential G_0, the screened Coulomb potential G_κ, and their first and second normal derivatives. Terms involving the potentials can be evaluated using the treecode as explained in the previous section, but terms involving the normal derivatives require a slight modification. For each potential G (either G_0 or G_κ), there are 16 terms that need to be evaluated; 1 term for the potential itself, 6 terms for the first partial derivatives $\partial_{x_m} G, \partial_{y_n} G$, and 9 terms for the second partial derivatives $\partial_{x_m} \partial_{y_n} G$, for $m, n = 1, 2, 3$. Each of the 16 terms can be evaluated as a modified form of Equation (16), obtained by applying the operator $\partial_{\mathbf{x}}^{\mathbf{k}_0} \partial_{\mathbf{y}}^{\mathbf{l}_0}$ to V_i and multiplying by a charge

p_i associated with particle $\mathbf{x}_i$,

$$p_i \partial_{\mathbf{x}}^{\mathbf{k}_0} \partial_{\mathbf{y}}^{\mathbf{l}_0} V_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} p_i q_j \partial_{\mathbf{x}}^{\mathbf{k}_0} \partial_{\mathbf{y}}^{\mathbf{l}_0} G(\mathbf{x}_i, \mathbf{x}_j) \tag{20a}$$

$$\approx \sum_{c \in N_i} \sum_{\mathbf{x}_j \in c} p_i q_j \partial_{\mathbf{x}}^{\mathbf{k}_0} \partial_{\mathbf{y}}^{\mathbf{l}_0} G(\mathbf{x}_i, \mathbf{x}_j) \tag{20b}$$

$$+ \sum_{c \in F_i} \sum_{||\mathbf{k}||=0}^{p} p_i (-1)^{||\mathbf{k}_0||} \frac{(\mathbf{k}+\mathbf{k}_0+\mathbf{l}_0)!}{\mathbf{k}!} a^{\mathbf{k}+\mathbf{k}_0+\mathbf{l}_0}(\mathbf{x}_i, \mathbf{x}_c)\, m_c^{\mathbf{k}}. \tag{20c}$$

Table 1 records the information needed to apply the treecode to compute the matrix-vector product, as derived from Equations (6), (8a)-(8b), (9), (10), (13a)-(13b). Column 1 is the index of the term; column 2 is the kernel K_i in which the term appears; column 3 is the potential appearing in kernel K_i; columns 4 and 5 are the required indices $\mathbf{k_0}, \mathbf{l}_0$; column 6 is the charge p_i related to particle $\mathbf{x}_i$; column 7 is the charge q_j related to particle $\mathbf{x}_j$. The Cartesian basis vectors are $\mathbf{e}_m, \mathbf{e}_n$ for $m, n = 1, 2, 3$.

Table 1: Information needed to apply the treecode to compute the matrix-vector product; $G = G_0, G_\kappa$.

term	kernel K_i	potential	index $\mathbf{k}_0$	index $\mathbf{l}_0$	charge p_i	charge q_j
1	K_1	G	$(0,0,0)$	$(0,0,0)$	1	$\frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu} A_j$
2-4	K_2	$\partial_{y_n} G$	$(0,0,0)$	$\mathbf{e}_n$	1	$\nu_n(\mathbf{x}_j)\phi_1(\mathbf{x}_j)A_j$
5-7	K_3	$\partial_{x_m} G$	$\mathbf{e}_m$	$(0,0,0)$	$\nu_m(\mathbf{x}_i)$	$\frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu} A_j$
8-16	K_4	$\partial_{x_m}\partial_{y_n} G$	$\mathbf{e}_m$	$\mathbf{e}_n$	$\nu_m(\mathbf{x}_i)$	$\nu_n(\mathbf{x}_j)\phi_1(\mathbf{x}_j)A_j$

3.4. Poisson equation

When the ionic concentration vanishes ($\kappa = 0\,\text{Å}^{-1}$), the PB equation reduces to the Poisson equation. In this case the system of integral equations (7a)-(7b) reduces to a single equation for the surface potential [22],

$$\frac{1}{2}(1+\varepsilon)\,\phi_1(\mathbf{x}) = (\varepsilon - 1)\int_\Gamma \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi_1(\mathbf{y}) dS_\mathbf{y} + \frac{1}{\varepsilon_1}\sum_{k=1}^{N_c} q_k G_0(\mathbf{x}, \mathbf{y}_k), \tag{21}$$

and the electrostatic solvation energy is

$$E_{sol} = \frac{1}{2}(\varepsilon - 1)\sum_{k=1}^{N_c} q_k \int_\Gamma \frac{\partial G_0(\mathbf{y}_k, \mathbf{y})}{\partial \nu_\mathbf{y}} \phi_1(\mathbf{y}) dS_\mathbf{y}. \tag{22}$$

In this case the centroid collocation scheme and treecode are applied as described above.

3.5. Coding details

The code was written in Fortran 90/95 and is available from the authors by request. Serial computations were performed on one processor of an 8-core workstation (each core is an Intel Xeon CPU at 2.83GHz with 2GB memory). We used the GMRES subroutine from Netlib [53], with zero initial guess and restart after every 10 steps. The linear system was scaled so that the diagonal element in each row is unity. Serial computations were compiled using ifort with option -fast and parallel computations were compiled using mpif90/gfortran with option -O3.

4. Test cases

We present results for two test cases. Case 1 is a sphere with an atomic charge at the center for which the exact solution of the PB equation was derived by Kirkwood [54]. In our computations the sphere has radius 50 Å, the charge has magnitude $q = 50\, e_c$, and the dielectric constant is $\varepsilon_1 = 1$ inside the sphere and $\varepsilon_2 = 40$ outside the sphere.

Case 2 is protein 1A63, the RNA binding domain of E. coli rho factor, with 2069 atoms [55]. Atom locations were obtained from the Protein Data Bank [56], and partial charges came from the CHARMM22 force field [57]. The dielectric constants are $\varepsilon_1 = 1$ in the molecular cavity and $\varepsilon_2 = 80$ in the solvent.

In both cases the Debye-Hückel constant is $\kappa = 0.1257\,\text{Å}^{-1}$ for the PB equation (corresponding to a physiological saline solution at room temperature) and $\kappa = 0\,\text{Å}^{-1}$ for the Poisson equation. In Case 1 we use a geodesic grid triangulation of the sphere obtained by successively dividing the faces of an icosahedron. In Case 2 we use the MSMS triangulation of the molecular surface [51, 52]. The MSMS code takes the atom locations as input, and outputs the vertices, normal vectors, and faces of the triangulation. A user-specified density parameter controls the number of vertices per Å^2 of surface area. The radius of the MSMS solvent probe sphere was 1.4 Å. The centroid normal vector required by the collocation scheme was obtained by averaging the MSMS vertex normal vectors and normalizing the result to unit length.

5. Numerical errors

There are two main types of numerical errors to consider in the context of the TABI solver; these are the *discretization error* and the *treecode approximation error*. There is also an error due to solving the linear system by GMRES iteration, but the GMRES tolerance τ is chosen so that this error is negligible compared to the others.

The discretization error arises from several sources including (a) the triangulation of the molecular surface, (b) applying centroid collocation to compute the surface integrals, and (c) omitting the singular term in the quadrature scheme. The size of the discretization error depends on the number of triangles N representing the molecular surface. In discussing the discretization error, it is assumed that the matrix-vector product is performed by direct summation.

The treecode approximation error is an additional error that arises from applying the treecode to compute the matrix-vector product. The size of the treecode approximation error depends on the order of Taylor approximation p and the MAC parameter θ. We will show that these parameters can be chosen so that the treecode approximation error is smaller than the discretization error.

We will present relative errors, expressed as percent (%). The discretization error is measured by comparing the direct sum numerical solution (ds) and the exact solution (ex),

$$e_{sol}^{ds} = \frac{|E_{sol}^{ds} - E_{sol}^{ex}|}{|E_{sol}^{ex}|}, \quad e_{\phi}^{ds} = \frac{||\{\phi\}^{ds} - \{\phi\}^{ex}||_\infty}{||\{\phi\}^{ex}||_\infty}, \quad e_{\phi_n}^{ds} = \frac{||\{\phi_n\}^{ds} - \{\phi_n\}^{ex}||_\infty}{||\{\phi_n\}^{ex}||_\infty}, \tag{23}$$

where E_{sol} is the electrostatic solvation energy, $\{\phi\}$ is the vector of surface potentials, and $\{\phi_n\}$ is the vector of surface potential normal derivatives. Note that in Case 1 (Kirkwood sphere), the exact solution is known analytically. In Case 2 (protein 1A63) the exact solution is not known, but as explained below, we will extrapolate the computed values to estimate the exact electrostatic solvation energy in Equation (23). The treecode approximation error is measured by comparing the treecode (tc) and direct sum (ds) numerical solutions,

$$e_{sol}^{tc} = \frac{|E_{sol}^{tc} - E_{sol}^{ds}|}{|E_{sol}^{ds}|}, \quad e_{\phi}^{tc} = \frac{||\{\phi\}^{tc} - \{\phi\}^{ds}||_\infty}{||\{\phi\}^{ds}||_\infty}, \quad e_{\phi_n}^{tc} = \frac{||\{\phi_n\}^{tc} - \{\phi_n\}^{ds}||_\infty}{||\{\phi_n\}^{ds}||_\infty}. \tag{24}$$

6. Results for Case 1 (Kirkwood sphere)

In this section we consider the Kirkwood sphere test case. We report first on the discretization error, then the effect of the treecode parameters, and finally we compare the treecode and direct sum in terms of error, CPU time, and memory usage.

6.1. Discretization error

Table 2 presents the discretization error for the Kirkwood sphere test case. The results are presented as a function of the number of faces in the triangulation N, for the PB and Poisson equations. The GMRES tolerance is $\tau = 10^{-6}$ and the matrix-vector product is computed by direct summation.

Table 2: Case 1 (Kirkwood sphere). Discretization error; PB and Poisson equations; results computed by direct summation; showing electrostatic solvation energy E_{sol}^{ds} and its discretization error e_{sol}^{ds}, and discretization error in surface potential e_{ϕ}^{ds}, normal derivative $e_{\phi_n}^{ds}$, as defined in Equation (23).

	Poisson-Boltzmann					Poisson			
N[1]	E_{sol}^{ds} (kcal/mol)	e_{sol}^{ds} (%)	e_{ϕ}^{ds} (%)	$e_{\phi_n}^{ds}$(%)	iters[2]	E_{sol}^{ds} (kcal/mol)	e_{sol}^{ds} (%)	e_{ϕ}^{ds} (%)	iters
320	-8410.47	1.658	11.047	3.980	4	-8253.77	1.971	4.319	3
1280	-8356.64	1.007	4.102	1.690	4	-8193.58	1.227	1.937	3
5120	-8318.13	0.542	3.723	0.764	4	-8148.64	0.672	0.913	3
20480	-8296.44	0.280	2.276	0.361	4	-8122.60	0.350	0.443	3
81920	-8285.04	0.142	1.241	0.175	4	-8108.71	0.179	0.218	3
327680	-8279.21	0.071	0.646	0.086	4	-8101.55	0.090	0.108	2
1310720	-8276.27	0.036	0.331	0.043	3	-8097.91	0.045	0.054	2
∞[3]	-8273.31					-8094.25			

[1]number of faces in triangulation
[2]number of GMRES iterations
[3]this row displays the exact electrostatic solvation energy E_{sol}^{ex}, which is known analytically

The results are summarized as follows. The computed electrostatic solvation energy E_{sol}^{ds} converges to the exact value at the rate $O(N^{-1/2})$; when N increases by a factor of four, the error decreases by a factor of approximately one-half, a standard result for this type of discretization and test case [28]. Despite the slow convergence rate, the discretization error is small for moderate values of N; for example with $N = 5120$, the energy error e_{sol}^{ds} is well below 1%. The energy is slightly lower for the PB equation than for the Poisson equation, but the errors are comparable in both cases.

Next consider the surface potential. The same convergence rate $O(N^{-1/2})$ as above is seen here also for the surface potential ϕ and its normal derivative ϕ_n. In the case of the PB equation, the normal derivative error $e_{\phi_n}^{ds}$ is smaller than the surface potential error e_{ϕ}^{ds}, as in previous computations using the present boundary integral formulation [22]. The surface potential error e_{ϕ}^{ds} is smaller for the Poisson equation than for the PB equation.

Finally note that the number of GMRES iterations is less than four in all cases in Table 2. This reflects the fact that the boundary integral formulation is well-conditioned, as known from prior work [22, 24].

Later in this section we will compare direct sum and treecode results, but next we examine the effect of the treecode parameters on the algorithm's performance.

6.2. Effect of treecode parameters

We applied the treecode to the Kirkwood sphere test case using the representative value $N = 81920$ for the triangulation. Figure 2 shows the effect of varying the approximation order p and

MAC parameter θ. The horizontal axis is the treecode approximation error in electrostatic solvation energy e_{sol}^{tc} (recall Equation (24)), and the vertical axis is the treecode CPU time in seconds. Each symbol in Figure 2 is the result of solving the PB equation (solid lines) or the Poisson equation (dashed lines) with the MAC parameter indicated in the legend ($\theta = 0.8, 0.5, 0.2$) and a given order ($p = 1:10$). Symbols with the same θ-value are connected by lines and the order p increases from right to left on each line.

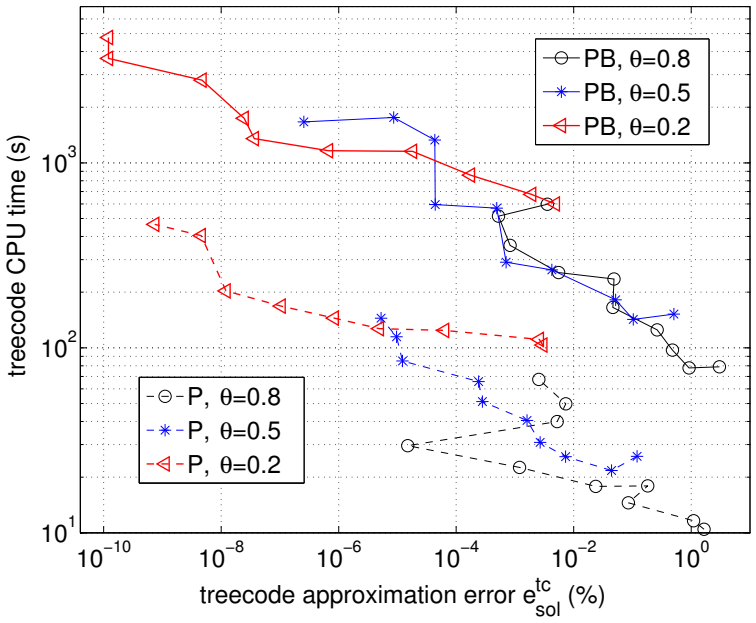

Figure 2: Case 1 (Kirkwood sphere). Effect of treecode parameters; CPU time (s) versus approximation error in electrostatic solvation energy e_{sol}^{tc} (%); PB equation (solid lines) and Poisson equation (dashed lines); $N = 81920$ faces, MAC parameter $\theta = 0.8, 0.5, 0.2$, order of Taylor approximation $p = 1:10$ (right to left on each line).

The following trends are observed. (1) The Poisson computation is approximately ten times faster than the PB computation. This is due to the fact that the Poisson integral equation (21) does not require solving for the surface potential normal derivative. (2) For a given MAC parameter θ, increasing the order p generally leads to a smaller error and larger CPU time, although occasionally it leads to a slightly larger error, a sign of non-monotone convergence in these multi-dimensional Taylor approximations. (3) Reducing the MAC parameter θ leads to a smaller error and larger CPU time. This can be understood by noting that reducing θ improves the rate of convergence of the Taylor approximation, but it also forces the treecode to descend to lower levels in the tree.

Figure 2 shows that choosing a large θ-value is generally more efficient for low accuracy, and a small θ-value is more efficient for high accuracy. In the remainder of this section we set the MAC parameter to $\theta = 0.5$ as a representative value. The following subsections compare the treecode and direct sum in terms of error, CPU time, and memory usage.

6.3. Comparison of treecode and direct sum: error

Figure 3 shows the error in electrostatic solvation energy versus the number of triangles N, comparing the discretization error e_{sol}^{ds} (ds, dashed lines) and treecode approximation error e_{sol}^{tc} (solid lines), for the PB and Poisson equations, with treecode order $p = 1, 3, 5, 7, 9$. The dashed lines have slope approximately $-\frac{1}{2}$, consistent with the discretization error results in Table 2. The treecode approximation error depends only weakly on the number of triangles N, and it can be

made smaller than the discretization error by increasing the order p. The PB and Poisson results are comparable. The error in the surface potential and normal derivative follow similar trends (not shown).

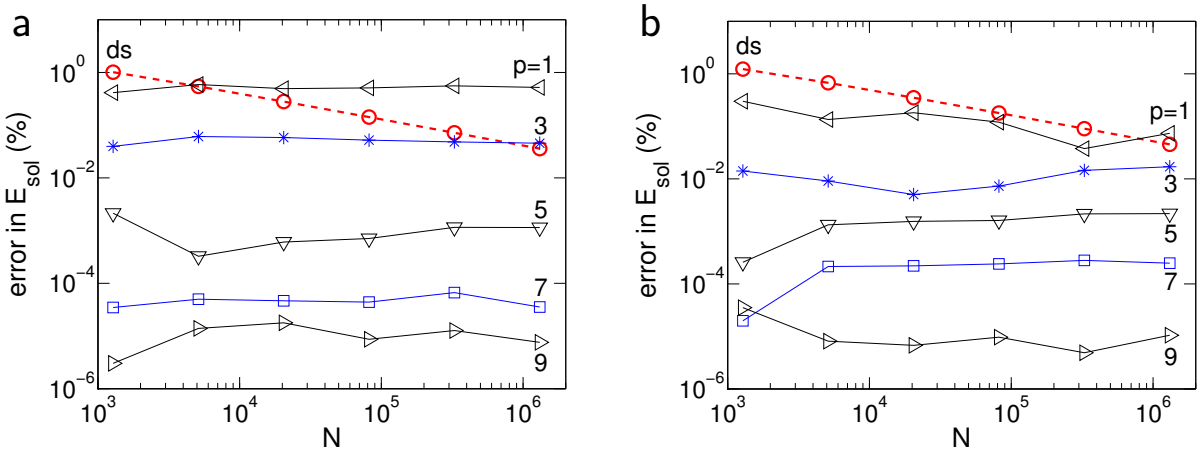

Figure 3: Case 1 (Kirkwood sphere). Error in electrostatic solvation energy versus number of triangles N; discretization error e_{sol}^{ds} (ds, dashed lines), treecode approximation error e_{sol}^{tc} (solid lines); (a) PB equation, (b) Poisson equation; treecode order $p = 1, 3, 5, 7, 9$, MAC parameter $\theta = 0.5$.

6.4. Comparison of treecode and direct sum: CPU time

Figure 4 shows the CPU time for direct sum (dashed lines) and the treecode (solid lines) versus the number of triangles N, for the PB and Poisson equations, with treecode order $p = 1, 3, 5, 7, 9$ (bottom to top). The direct sum CPU time increases at the rate $O(N^2)$, while the treecode CPU time increases at the rate $O(N \log N)$. For a given value of N, the treecode CPU time increases with the order p, but depending on the required accuracy, significant speedup is achieved compared to direct sum. As in Figure 2, solving the Poisson equation is faster than solving the PB equation.

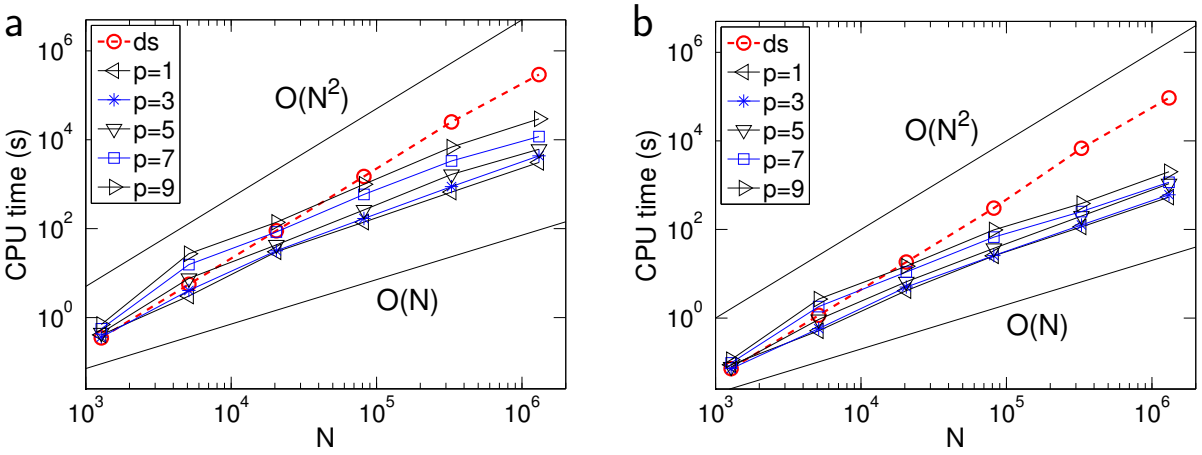

Figure 4: Case 1 (Kirkwood sphere). CPU time for direct sum (dashed lines) and treecode (solid lines) versus number of triangles N; (a) PB equation, (b) Poisson equation; treecode order $p = 1, 3, 5, 7, 9$ (bottom to top), MAC parameter $\theta = 0.5$.

6.5. Comparison of treecode and direct sum: memory usage

Figure 5 shows the memory usage for direct sum (dashed lines) and treecode (solid lines) versus the number of triangles N, for the PB and Poisson equations, with treecode order $p = 1, 3, 5, 7, 9$

(bottom to top). The treecode uses more memory than direct sum, and the memory usage increases with the order p, but the treecode and direct sum memory usage are both $O(N)$. The Poisson equation requires less memory than the PB equation.

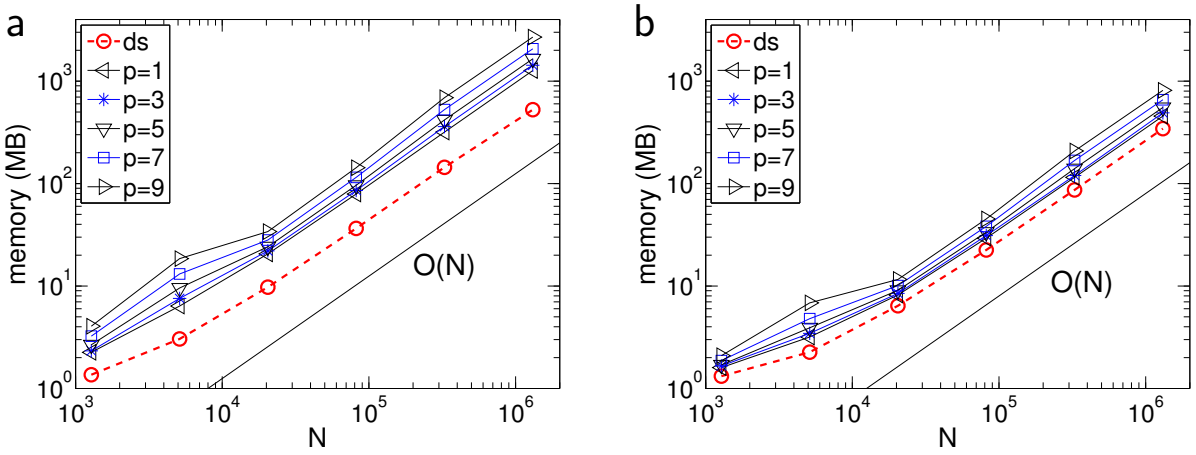

Figure 5: Case 1 (Kirkwood sphere). Memory usage for direct sum (dashed lines) and treecode (solid lines) versus number of triangles N; (a) PB equation, (b) Poisson equation; treecode order $p = 1, 3, 5, 7, 9$ (bottom to top), MAC parameter $\theta = 0.5$.

7. Results for Case 2 (protein 1A63)

Next we apply the TABI solver to protein 1A63. The molecular surface is triangulated by MSMS [51, 52], with atom locations from the Protein Data Bank [56] and partial charges from the CHARMM22 force field [57]. MSMS has a user-specified density parameter giving the number of vertices per Å^2 in the triangulation. Figure 6 displays the MSMS triangulation for protein 1A63 with density $= 1$ Å^{-2} and density $= 5$ Å^{-2}. MSMS produces a non-uniform adapted triangulation which becomes smoother as the vertex density increases. Note that MSMS may produce some extremely small triangles which can lead to numerical difficulties; in the present case, any triangle with area less than 10^{-6}Å^2 is removed from the computation. The treecode uses order $p = 3$, and MAC parameter $\theta = 0.8$ for the PB equation and $\theta = 0.5$ for the Poisson equation. The GMRES tolerance is $\tau = 10^{-4}$. These are representative parameter values chosen to ensure that the treecode approximation error and GMRES iteration error are smaller than the direct sum discretization error.

We will compare TABI results with those obtained using the grid-based APBS code (version 1.4.0) [18, 58]. To carry out the APBS simulation, since protein 1A63 is longer in one direction, the protein is placed in a rectangular box of dimensions 66 Å $\times$ 41 Å $\times$ 42 Å. The box is discretized by a Cartesian grid with N_g grid points, where $N_g = N_1 \times N_2^2$, and we denote the maximum grid spacing by h_{max}. APBS has several options for treating the interface, far-field boundary conditions, and atomic charges, and we chose standard parameter values (bcfl = mdh, chgm = spl2, nlev = 4, sdens = 10, srad = 1.4, srfm = mol, swin = 0).

7.1. Particle clusters

Before proceeding we note that the particle clusters in a treecode are typically uniform cubes at each level [47]. We followed that approach for the Kirkwood sphere test case above, but in the case of protein 1A63, we will instead use adapted rectangular boxes obtained by shrinking the clusters around the particles they contain [48–50]. Figure 7 illustrates the idea for a two-dimensional

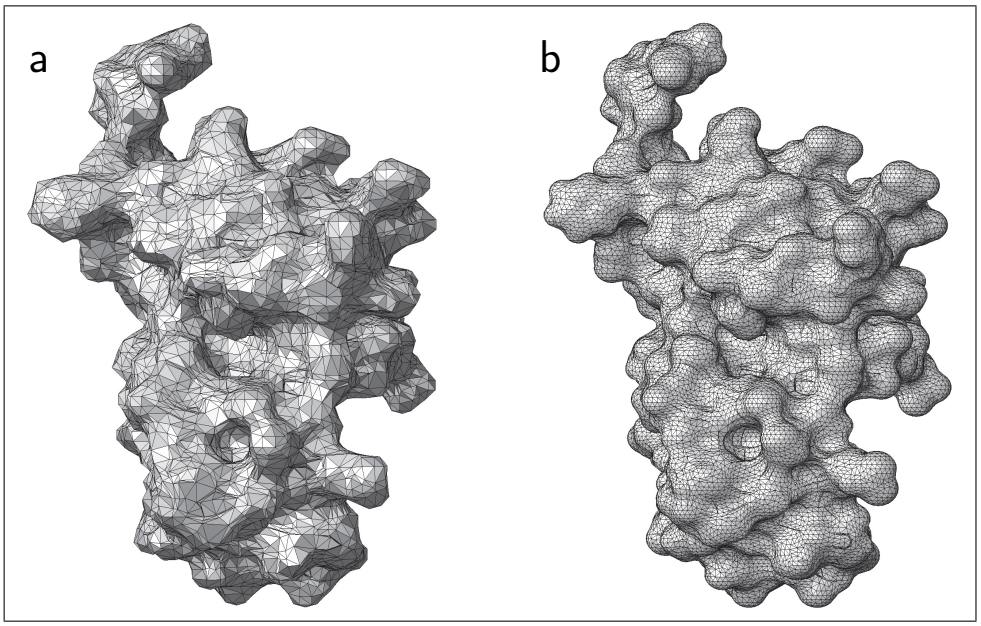

Figure 6: Case 2 (protein 1A63). MSMS triangulation of molecular surface [51, 52]; (a) density = 1 Å^{-2}, $N = 20264$ faces, (b) density = 5 Å^{-2}, $N = 70018$ faces.

analog of a molecular surface. In comparison with uniform clusters, the adapted clusters have smaller radius and provide a better description of the molecular surface.

To demonstrate the effect of using adapted clusters, we applied the TABI solver to protein 1A63 with MSMS density = 10 Å^{-2} and $N = 132196$ faces. Table 3 displays the resulting electrostatic solvation energy E_{sol} computed by direct sum (ds), and by the treecode with uniform clusters ($tc1$) and adapted clusters ($tc2$), followed by the discretization error e^{ds}_{sol} and treecode approximation error e^{tc}_{sol}, and the CPU run time. In computing the discretization error, the exact value is estimated by an extrapolation process described in the next subsection.

Results in Table 3 are shown for the PB equation with MAC parameter $\theta = 0.8$, and the Poisson equation with $\theta = 0.5$. In all cases, the treecode approximation error is smaller than the discretization error, and the treecode is faster than direct sum. In using adapted clusters in comparison with uniform clusters, the CPU run time is reduced, by 15% for the PB equation and 23% for the Poisson equation. This can be explained by noting that the adapted clusters have smaller radius than the uniform clusters, and as a result the MAC criterion is satisfied at higher levels in the tree. Hence we will use adapted particle clusters in the remainder of this work.

Table 3: Case 2 (protein 1A63). Effect of particle clusters; PB and Poisson equations; MSMS density = 10 Å^{-2}, $N = 132196$; showing electrostatic solvation energy E_{sol} computed by direct sum (ds), treecode with uniform clusters ($tc1$), adapted clusters ($tc2$); discretization error e^{ds}_{sol}, treecode approximation error e^{tc}_{sol}, CPU time; order $p = 3$, MAC parameter θ as indicated.

	E_{sol} (kcal/mol)			error (%)			CPU (s)		
	ds	$tc1$	$tc2$	e^{ds}_{sol}	e^{tc1}_{sol}	e^{tc2}_{sol}	ds	$tc1$	$tc2$
PB, $\theta = 0.8$	-2404.072	-2403.837	-2405.113	1.239	0.00978	0.04329	10768	434	368
Poisson, $\theta = 0.5$	-2399.471	-2399.388	-2399.480	1.295	0.00346	0.00037	3761	220	169

In the next three subsections we discuss in more detail the error in electrostatic solvation energy, CPU time, and memory usage. Results are presented in Table 4.

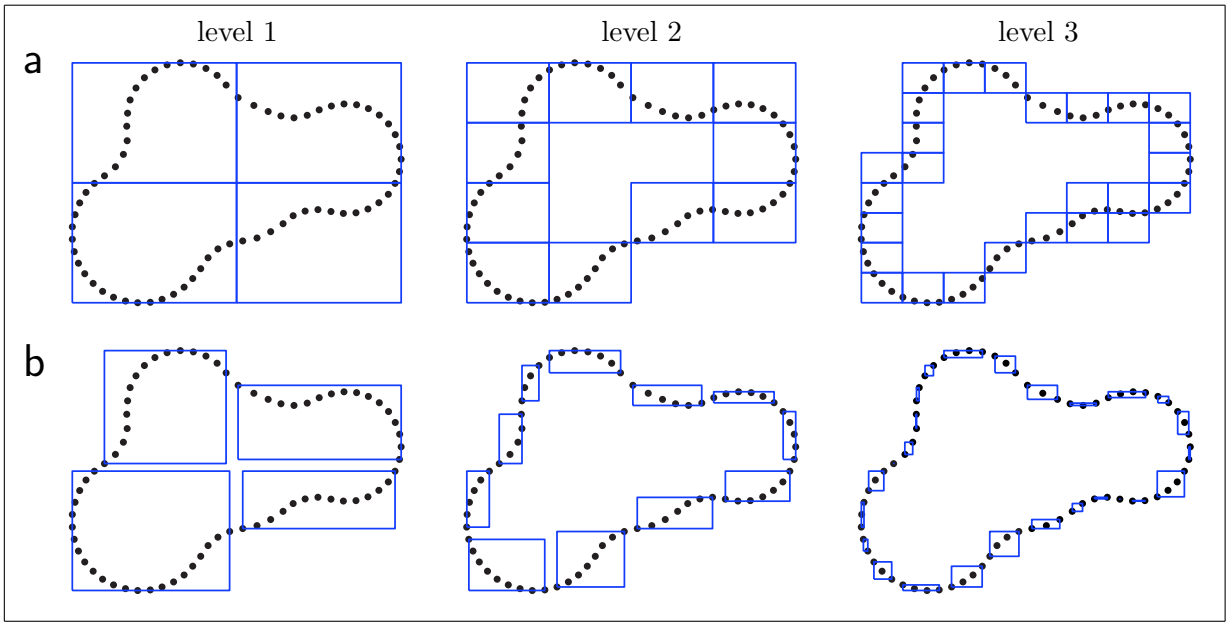

Figure 7: Treecode particle clusters, two-dimensional analog of a molecular surface. (a) uniform, (b) adapted.

7.2. Error in electrostatic solvation energy

The TABI Poisson-Boltzmann results appear in the top of Table 4. The first two columns give the MSMS density and number of faces in the triangulation N. The next two columns give the electrostatic solvation energy E_{sol} computed by direct sum (ds) and treecode (tc). We find empirically that the direct sum values E_{sol}^{ds} converge at the rate $O(N^{-1})$; this is supported by Figure 8a in which these values are plotted versus N^{-1}, showing that the data points asymptote to a straight line as $N^{-1} \to 0$. Using this observation, we performed linear extrapolation to the limit $N^{-1} \to 0$, obtaining the value $E_{sol}^{ex} = -2374.64\,\text{kcal/mol}$, an estimate of the exact energy, which appears on the row labelled ∞ in Table 4. Using this value we computed the discretization error e_{sol}^{ds} (recall Equation (23)), and the results in Table 4 support the conclusion that it converges to zero at the rate $O(N^{-1})$. This is faster than the rate $O(N^{-1/2})$ obtained for the geodesic grid triangulation of the Kirkwood sphere in Case 1. The faster convergence seen here is attributed to the non-uniform adaptive nature of the MSMS triangulation of the protein molecular surface; there are two ways to interpret this, (1) for a given number of faces N, the MSMS triangulation yields a smaller error than a uniform triangulation, (2) for a given level of error, MSMS requires a smaller number of faces N than a uniform triangulation.

Returning to the TABI-PB results in Table 4, we see that the treecode values of electrostatic solvation energy are in good agreement with the direct sum values. In all cases, the treecode approximation error e_{sol}^{tc} remains smaller than the direct sum discretization error e_{sol}^{ds}. For example with density $= 10\,\text{Å}^{-2}$, the direct sum discretization error is $e_{sol}^{ds} = 1.239\%$ and the treecode approximation error is $e_{sol}^{tc} = 0.0433\%$.

The next portion of Table 4 displays TABI results for the Poisson equation. In this case the estimate of the exact electrostatic solvation energy is $E_{sol}^{ex} = -2368.79\,\text{kcal/mol}$, approximately 5.8 kcal/mol higher than for the PB equation. Otherwise, the error trends for the PB and Poisson equations are similar; the only difference is that the treecode approximation error e_{sol}^{tc} is smaller for the Poisson equation, due to the smaller MAC parameter value used in this case.

Proceeding to the APBS results in Table 4, the first three columns display the maximum grid spacing h_{max}, grid dimensions N_g, and electrostatic solvation energy E_{sol}. We find empirically that the computed values E_{sol} converge at the rate $O(h_{max})$, as supported by Figure 8b. In the

Table 4: Case 2 (protein 1A63). TABI and APBS results; PB and Poisson equations; showing electrostatic solvation energy E_{sol}, and error, CPU time, memory usage; TABI columns show MSMS density, E_{sol} values computed by direct sum (ds) and treecode (tc); discretization error e_{sol}^{ds}, treecode approximation error e_{sol}^{tc}; treecode order $p = 3$, MAC parameter $\theta = 0.8$ (PB), $\theta = 0.5$ (P); APBS columns show maximum grid spacing h_{max}, grid dimensions N_g.

TABI, Poisson-Boltzmann		E_{sol} (kcal/mol)		error (%)		CPU (s)		iters[2]		memory (MB)	
density (Å^{-2})	N[1]	ds	tc	e_{sol}^{ds}	e_{sol}^{tc}	ds	tc	ds	tc	ds	tc
1	20264	-2913.46	-2914.22	22.691	0.0260	535	87	25	25	10	23
2	30358	-2531.67	-2532.65	6.613	0.0385	1059	130	22	22	14	36
5	70018	-2440.05	-2440.80	2.755	0.0717	3371	205	13	13	31	82
10	132196	-2404.07	-2405.11	1.239	0.0433	10768	368	11	11	57	149
20	265000	-2390.12	-2392.51	0.652	0.0997	39187	812	10	11	113	309
40	536886	-2382.28	-2385.74	0.322	0.1450	178418	1763	11	11	227	600
	∞[3]	-2374.64									
TABI, Poisson											
1	20264	-2908.49	-2908.47	22.784	0.0007	171	41	28	28	7	9
2	30358	-2526.42	-2526.51	6.655	0.0036	273	51	20	20	10	14
5	70018	-2436.33	-2436.33	2.851	0.0003	1089	83	13	13	22	31
10	132196	-2399.47	-2399.48	1.295	0.0004	3761	169	12	12	40	56
20	265000	-2385.04	-2385.01	0.686	0.0013	14200	322	11	11	80	114
40	536886	-2376.81	-2376.77	0.339	0.0017	61098	724	11	11	160	223
	∞	-2368.79									

APBS, Poisson-Boltzmann						
h_{max} (Å)	N_g	E_{sol} (kcal/mol)	error (%)	CPU (s)		memory (MB)
1.63	65×33^2	-2455.15	4.449	6		93
0.812	97×65^2	-2559.61	8.893	19		164
0.547	129×97^2	-2477.78	5.411	72		340
0.263	257×161^2	-2411.03	2.572	292		1565
0.131	513×321^2	-2380.69	1.281	1983		11486
	∞	-2350.58				
APBS, Poisson						
1.63	65×33^2	-2439.01	4.066	6		82
0.812	97×65^2	-2550.60	8.827	16		157
0.547	129×97^2	-2469.49	5.367	65		333
0.263	257×161^2	-2404.72	2.603	247		1559
0.131	513×321^2	-2374.10	1.297	1647		11479
	∞	-2343.71				

[1]number of faces in triangulation
[2]number of GMRES iterations
[3]rows labelled ∞ display estimates of exact energy E_{sol}^{ex} obtained by extrapolation as in Figure 8

same spirit as above, we performed linear extrapolation to the limit $h_{max} \to 0$, obtaining the value $E_{sol} = -2350.58$ kcal/mol for the PB equation, which appears on the appropriate row labelled ∞ in Table 4. This is 24 kcal/mol higher than the TABI value, a discrepancy of 1%, possibly due to the different treatment of the molecular surface or the far-field boundary condition in the two codes. The next column in Table 4 shows that the error converges to zero at the rate $O(h_{max})$, a check on the self-consistency of the extrapolation. The error trends for the PB and Poisson equations are again similar. The energy for the Poisson equation is $E_{sol} = -2343.71$ kcal/mol, approximately 6.9 kcal/mol higher than for the PB equation.

7.3. CPU time

Proceeding to the CPU time results in Table 4 for the TABI solver, the direct sum CPU time is $O(N^2)$ and the treecode CPU time is $O(N \log N)$. Hence the treecode is significantly faster when N is large; for example, the PB computation with density $= 10$ Å^{-2} and $N = 132196$ took 10768 s ≈ 3 hours by direct sum and 368 s ≈ 6.1 minutes by the treecode. The number of GMRES

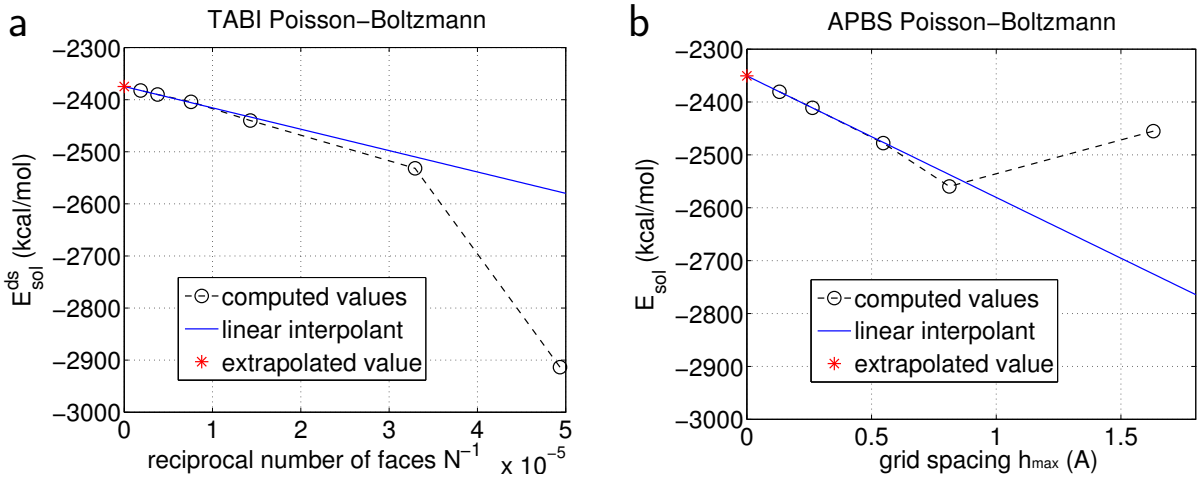

Figure 8: Case 2 (protein 1A63). Estimation of exact PB electrostatic solvation energy; computed values of E_{sol} from Table 4 ($\circ$, dashed lines); linear interpolant (solid lines); (a) TABI direct sum values E^{ds}_{sol} versus reciprocal number of faces N^{-1}, (b) APBS values E_{sol} versus grid spacing h_{max} (Å); extrapolated values ($*$) appear in Table 4 on rows labelled ∞.

iterations is modest and decreases as N becomes larger, presumably due to the better resolution of the molecular surface and the use of a well-conditioned boundary integral formulation [22, 24]. The treecode CPU time for the Poisson equation is 40-50% of the CPU time for the PB equation; for example in the same case with density $= 10\,\text{Å}^{-2}$, the CPU time for the Poisson equation is 169 s ≈ 2.8 minutes.

In the case of APBS, the CPU time increases at a rate somewhat less than $O(N_g)$ as the grid is refined. Except for the coarsest grid in Table 4, the CPU time for the Poisson equation is 80-90% of the CPU time for the PB equation.

Next in Figure 9a we compare TABI and APBS by plotting the CPU time versus error in electrostatic solvation energy for the PB and Poisson equations. The APBS error is taken directly from Table 4, and the TABI error is computed from the data in Table 4 as $|E^{tc}_{sol} - E^{ex}_{sol}|/|E^{ex}_{sol}|$. For the PB equation, APBS is faster for errors greater than 3% and TABI is faster for errors less than 3%. For the Poisson equation, the cross-over is at error 6%. Hence TABI is more efficient when higher accuracy is required. For example, the TABI simulation with density $= 10\,\text{Å}^{-2}$ and the APBS simulation with $h_{max} = 0.131$ Å both have errors around 1.3%, but TABI is 5.4 times faster for the PB equation and 9.7 times faster for the Poisson equation.

7.4. Memory usage

The final columns in Table 4 display the memory usage. The TABI memory usage is $O(N)$ for both direct sum and the treecode. For the PB equation the treecode uses 2-3 times as much memory as direct sum, and for the Poisson equation the treecode uses less than 2 times as much memory as direct sum. The APBS memory usage is $O(N_g)$.

Figure 9b compares TABI and APBS by plotting the memory usage versus error in electrostatic solvation energy for the PB and Poisson equations. The data is taken from Table 4. We find that TABI uses less memory than APBS for comparable levels of accuracy. For example, with errors around 1.3%, APBS with $h_{max} = 0.131$ Å uses 11486 MB for the PB equation and 11479 MB for the Poisson equation, while TABI with density $= 10\,\text{Å}^{-2}$ uses 149 MB for the PB equation and 56 MB for the Poisson equation.

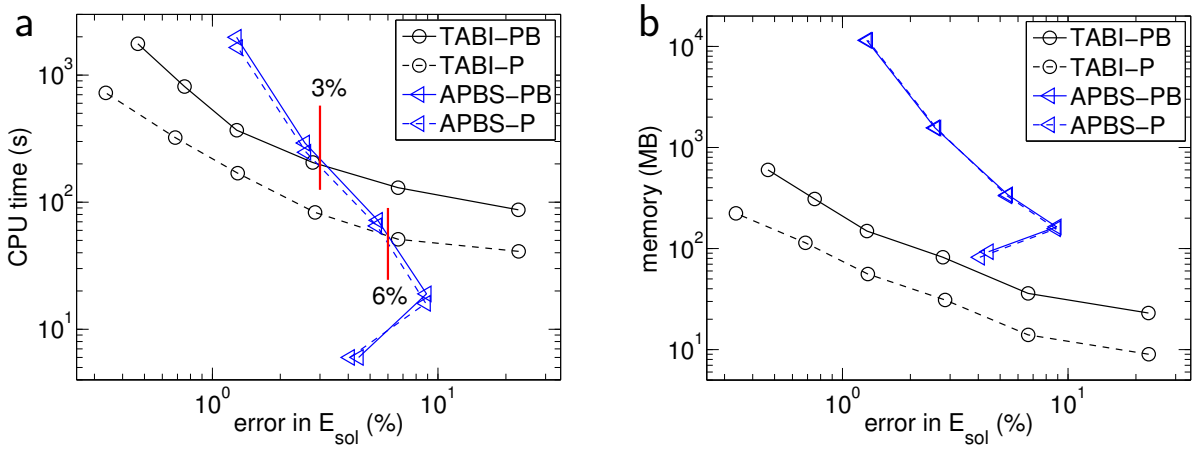

Figure 9: Case 2 (protein 1A63). Comparison of TABI and APBS; (a) CPU time, (b) memory usage, plotted versus error in electrostatic solvation energy; TABI (○), APBS (◁), PB equation (solid lines), Poisson equation (dashed lines); data from Table 4; vertical lines in (a) indicate cross-over error below which TABI is faster than APBS.

7.5. Visualization of surface potential

Figure 10 displays the surface potential for protein 1A63 computed by TABI with density $= 10$ Å^{-2} for the PB and Poisson equations. The plots were generated using VMD [59]. The regions of positive and negative potential correspond well for both equations, but the color intensity is slightly lower for the PB equation due to the screening induced by the dissolved ions. The difference potential $\phi_{PB} - \phi_P$ is shown in Figure 10c, with a smaller range in the color bar to emphasize the screening effect. These types of plots are used in the study of protein structure and binding affinity [1].

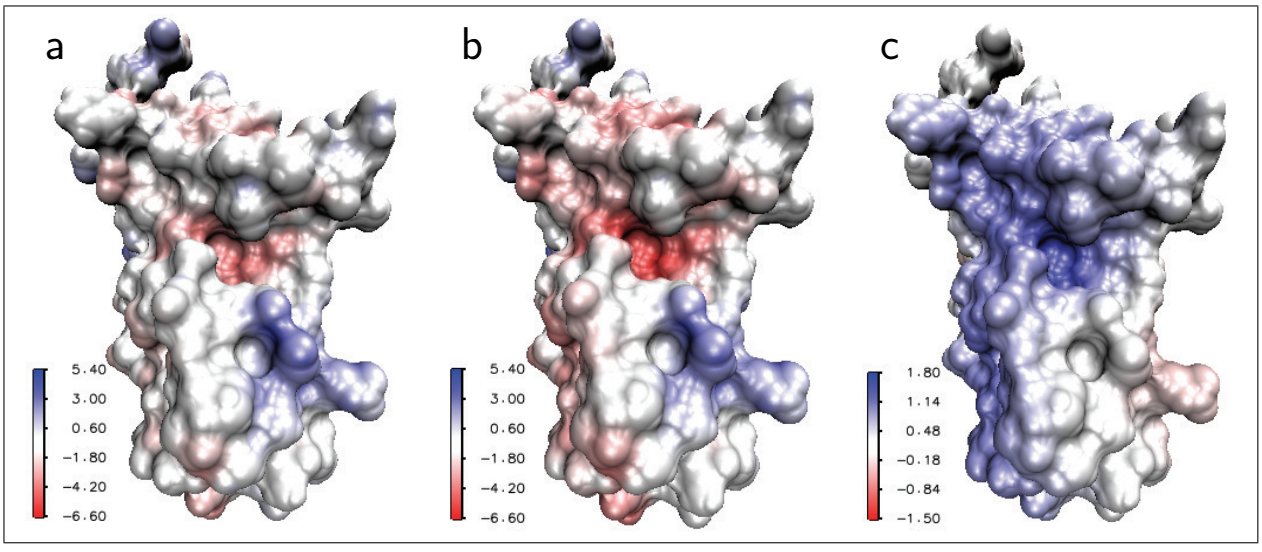

Figure 10: Case 2 (protein 1A63). Visualization of surface potential computed by TABI with density $= 10$ Å^{-2}; color bar (online) in units of kcal/mol-e_c; (a) PB equation; (b) Poisson equation, (c) difference potential $\phi_{PB} - \phi_P$, note smaller range in color bar.

7.6. Parallel simulations

Parallel treecode simulations are a topic of ongoing research e.g. see [60–63]. Here we present parallel TABI simulations for protein 1A63 with density $= 10\,\text{Å}^{-2}$ and $N = 132196$, yielding an error of around 1.3% in electrostatic solvation energy. In this case the memory usage is small enough to permit application of a simple replicated data algorithm in which each processor has a copy of all the data needed for concurrent computations [64, 65]. Larger TABI simulations requiring a distributed memory approach are reserved for future work.

The replicated data algorithm is based on the following considerations. The TABI solver computes the matrix-vector product using the treecode, and it computes the source terms (Equation (11)) and electrostatic solvation energy (Equation (14)) by direct sum. In each case the code loops overs the particles, but each particle can be treated as an independent computation. Hence the particle array is divided into P segments of length N/P, where P is the number of processors, and the segments are processed concurrently. The pseudocode is shown in Table 5. Communication is handled by MPI [66].

Table 5: Pseudocode for parallel TABI solver using replicated data algorithm.

```
on main processor
    read protein data
    call MSMS to generate triangulation
    copy protein data and triangulation to all other processors
on each processor
    build local copy of tree
    compute assigned segment of source terms by direct sum
        copy result to all other processors
    set initial guess for GMRES iteration
    compute assigned segment of matrix-vector product by treecode
        copy result to all other processors
    test for GMRES convergence
        if no, go to step 10 for next iteration
        if yes, go to step 15
    compute assigned segment of electrostatic solvation energy by direct sum
        copy result to main processor
on main processor
    add segments of electrostatic solvation energy and output result
```

Table 6 displays the CPU time (t_P), speedup (t_1/t_P), and parallel efficiency (speedup/P), for $P = 1, 2, 4, 8$ processors. Results are shown for the total computation and one matrix-vector product. The total computation time includes MSMS triangulation, tree building, GMRES iteration, and computing electrostatic solvation energy. Note that the total CPU time for one processor in Table 6 (PB=799.3 s, P=324.4 s) is higher than the serial CPU time in Table 4 (PB=368 s, P=169 s); this is attributed to the difference in compilers (ifort, gfortran) as well as overhead in running the parallel code with one processor. For the PB equation with eight processors, the total CPU time is reduced to 123.7 s, with speedup 6.46 and parallel efficiency 80.9%. For the Poisson equation with eight processors, the total CPU time is 51.0 s, around 40% of the CPU time for the PB equation, with comparable speedup and parallel efficiency. Results for one matrix-vector product are slightly better, with speedup 6.64 and 6.70, and parallel efficiency 83.0% and 83.8% for the PB and Poisson equations, respectively.

Table 6: Case 2 (protein 1A63). Performance of parallel TABI solver showing CPU time, speedup, parallel efficiency; PB and Poisson equations; total computation and one matrix-vector product; number of processors $P = 1, 2, 4, 8$; MSMS density $= 10\,\text{Å}^{-2}$, $N = 132196$ faces; treecode order $p = 3$, MAC parameter $\theta = 0.8$ (PB), $\theta = 0.5$ (P); error in electrostatic solvation energy $\approx 1.3\%$.

	total computation			one matrix-vector product		
P	CPU (s)	speedup	parallel efficiency (%)	CPU (s)	speedup	parallel efficiency (%)
Poisson-Boltzmann						
1	799.3	1.00	100.0	69.8	1.00	100.0
2	410.0	1.95	97.5	35.7	1.96	97.8
4	223.8	3.57	89.3	19.4	3.59	89.9
8	123.7	6.46	80.9	10.5	6.64	83.0
Poisson						
1	324.4	1.00	100.0	25.9	1.00	100.0
2	166.7	1.95	97.3	13.2	1.96	98.2
4	92.6	3.50	87.6	7.2	3.57	89.3
8	51.0	6.36	79.5	3.9	6.70	83.8

8. Conclusions

We presented a treecode-accelerated boundary integral (TABI) solver for electrostatics of solvated biomolecules described by the linear Poisson-Boltzmann implicit solvent model. The method employs a well-conditioned boundary integral formulation for the electrostatic potential and its normal derivative on the molecular surface [22]. The surface is triangulated and the integral equations are discretized by centroid collocation with the singular term omitted [46]. The linear system is solved by GMRES iteration [44] and the matrix-vector product is carried out by a Cartesian treecode which reduces the cost from $O(N^2)$ to $O(N \log N)$, where N is the number of faces in the triangulation [47, 50].

The TABI solver was applied to compute the electrostatic solvation energy in two cases, the Kirkwood sphere [54] and protein 1A63 with 2069 atoms [55]. The sphere was triangulated using an icosahedral geodesic grid and the protein surface was triangulated by MSMS [51]. We found that the discretization error is $O(N^{-1/2})$ for the sphere and $O(N^{-1})$ for the protein, where the faster convergence in the latter case is attributed to the adaptive nature of the MSMS triangulation. We showed that the treecode approximation error can be made smaller than the discretization error by a suitable choice of treecode parameters. TABI simulations for the PB and Poisson equations have similar trends in error, CPU time, and memory usage, but the code is faster and uses less memory for the Poisson equation.

We compared two versions of the treecode, one with uniform clusters and one with non-uniform clusters adapted to the molecular surface. The version with adapted clusters has better performance; the treecode approximation error remains smaller than the discretization error, and the CPU run time is reduced. The adapted cluster technique is a novel feature of the TABI solver and it may be especially effective for large biomolecules with complex geometry.

We also applied the grid-based APBS code [18] to protein 1A63. In the case of the PB equation, we found that APBS is faster than TABI for errors greater than 3% and TABI is faster for errors less than 3%. In the case of the Poisson equation, a similar cross-over occurs at error 6%. For comparable accuracy, TABI uses less memory than APBS. These results give a performance snapshot and further improvement can be expected in both grid-based and boundary integral PB solvers.

Finally, we presented parallel TABI simulations using a replicated data algorithm for protein 1A63 with density $= 10\,\text{Å}^{-2}$ and $N = 132196$, yielding 1.3% error in electrostatic solvation energy. With eight processors, TABI achieved parallel efficiency around 80%, requiring 123.7 s for the PB

equation and 51.0 s for the Poisson equation.

In summary, the TABI solver described here exhibits good serial and parallel performance combined with relatively simple implementation, efficient memory usage, and geometric adaptability. Hence it offers an attractive option for computing electrostatics of solvated biomolecules. Directions for future study include higher order quadrature schemes [22, 24, 32, 45, 67], alternative representations of the molecular surface [68–72], and PB simulations with quantum mechanical models of the solute [73–76].

Acknowledgements

The authors thank Benzhuo Lu for helpful discussions and the reviewers for suggestions that improved the manuscript. The work was supported by NSF grant DMS-0915057.

[1] B. Honig, A. Nicholls, Classical electrostatics in biology and chemistry, Science 268 (1995) 1144–1149.

[2] Z. Zhang, S. Witham, E. Alexov, On the role of electrostatics in protein-protein interactions, Phys. Biol. 8 (2011) 035001.

[3] B. Roux, T. Simonson, Implicit solvent models, Biophys. Chem. 78 (1999) 1–20.

[4] M. Feig, C.L. Brooks III, Recent advances in the development and application of implicit solvent models in biomolecule simulations, Curr. Opin. Struct. Biol. 14 (2004) 217–224.

[5] I. Klapper, R. Hagstrom, R. Fine, K. Sharp, B. Honig, Focusing of electric fields in the active site of Cu-Zn superoxide dismutase: effects of ionic strength and amino-acid modification, Proteins: Struct., Funct., Genet. 1 (1986) 47–59.

[6] M.E. Davis, J.A. McCammon, Electrostatics in biomolecular structure and dynamics, Chem. Rev. 90 (1990) 509–521.

[7] F. Fogolari, A. Brigo, H. Molinari, The Poisson-Boltzmann equation for biomolecular electrostatics: a tool for structural biology, J. Mol. Recognit. 15 (2002) 377-392.

[8] N.A. Baker, Poisson-Boltzmann methods for biomolecular electrostatics, Methods Enzymol. 383 (2004) 94–118.

[9] K.M. Callenberg, O.P. Choudhary, G.L. de Forest, D.W. Gohara, N.A. Baker, M. Grabe, APBSmem: A graphical interface for electrostatic calculations at the membrane, PLoS ONE 5 (2010) e12722.

[10] D.A. Beard, T. Schlick, Modeling salt-mediated electrostatics of macromolecules: The Discrete Surface Charge Optimization algorithm and its application to the nucleosome, Biopolymers 58 (2001) 106–115.

[11] N.A. Baker, Improving implicit solvent simulations: a Poisson-centric view, Curr. Opin. Struct. Biol. 15 (2005) 137–143.

[12] J.H. Chen, C.L. Brooks III, J. Khandogin, Recent advances in implicit solvent-based methods for biomolecular simulations, Curr. Opin. Struct. Biol. 18 (2008) 140–148.

[13] F.M. Richards, Areas, volumes, packing, and protein structure, Annu. Rev. Biophys. Bioeng. 6 (1977) 151–176.

[14] M.L. Connolly, Molecular surface triangulation, J. Appl. Crystallogr. 18 (1985) 499–505.

[15] M.E. Davis, J.D. Madura, B.A. Luty, J.A. McCammon, Electrostatics and diffusion of molecules in solution: simulations with the University of Houston Brownian dynamics program Comput. Phys. Commun. 62 (1991) 187–197.

[16] W. Im, D. Beglov, B. Roux, Continuum solvation model: Computation of electrostatic forces from numerical solutions to the Poisson-Boltzmann equation, Comput. Phys. Commun. 111 (1998) 59–75.

[17] N. Baker, M. Holst, F. Wang, Adaptive multilevel finite element solution of the Poisson-Boltzmann equation II. Refinement at solvent-accessible surfaces in biomolecular systems, J. Comput. Chem. 21 (2000) 1343–1352.

[18] N.A. Baker, D. Sept, S. Joseph, M.J. Holst, J.A. McCammon, Electrostatics of nanosystems: application to microtubules and the ribosome, Proc. Natl. Acad. Sci. USA 98 (2001) 10037–10041.

[19] R. Luo, L. David, M.K. Gilson, Accelerated Poisson-Boltzmann calculations for static and dynamic systems, J. Comput. Chem. 23 (2002) 1244–1253.

[20] A.H. Boschitsch, M.O. Fenley, A fast and robust Poisson-Boltzmann solver based on adaptive Cartesian grids, J. Chem. Theory Comput. 7 (2011) 1524–1540.

[21] D. Chen, Z. Chen, C.J. Chen, W.H. Geng, G.W. Wei, MIBPB: A software package for electrostatic analysis, J. Comput. Chem. 32 (2011) 756–770.

[22] A. Juffer, E. Botta, B. van Keulen, A. van der Ploeg, H. Berendsen, The electric potential of a macromolecule in a solvent: a fundamental approach, J. Comput. Phys. 97 (1991) 144–171.

[23] R. Bharadwaj, A. Windemuth, S. Sridharan, B. Honig, A. Nicholls, The fast multipole boundary element method for molecular electrostatics: An optimal approach for large systems, J. Comput. Chem. 16 (1995) 898–913.

[24] J. Liang, S. Subramaniam, Computation of molecular electrostatics with boundary element methods, Biophys. J. 73 (1997) 1830–1841.

[25] A.H. Boschitsch, M.O. Fenley, H.-X. Zhou, Fast boundary element method for the linear Poisson-Boltzmann equation, J. Phys. Chem. B 106 (2002) 2741–2754.

[26] M.D. Altman, J.P. Bardhan, B. Tidor, J.K. White, FFTSVD: A fast multiscale boundary-element method solver suitable for bio-MEMS and biomolecule simulation, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 25 (2006) 274-284.

[27] S. Grandison, R. Penfold, J.-M. Vanden-Broeck, A rapid boundary integral equation technique for protein electrostatics, J. Comput. Phys. 224 (2007) 663-680.

[28] B.Z. Lu, X. Cheng, J.A. McCammon, "New-version-fast-multipole-method" accelerated electrostatic calculations in biomolecular systems, J. Comput. Phys. 226 (2007) 1348–1366.

[29] J.P. Bardhan, Numerical solution of boundary-integral equations for molecular electrostatics, J. Chem. Phys. 130 (2009) 094102.

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

[31] B.Z. Lu, X. Cheng, J.F. Huang, J.A. McCammon, AFMPB: an adaptive fast multipole Poisson-Boltzmann solver for calculating electrostatics in biomolecular systems, Comput. Phys. Commun. 181 (2010) 1150–1160.

[32] C. Bajaj, S.-C. Chen, A. Rand, An efficient higher-order fast multipole boundary element solution for Poisson-Boltzmann-based molecular electrostatics, SIAM J. Sci. Comput. 33 (2011) 826–848.

[33] R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada, Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUS and a billion unknowns, Comput. Phys. Commun. 182 (2011) 1272–1283.

[34] B. Zhang, B.Z. Lu, X. Cheng, J.F. Huang, N.P. Pitsianis, X. Sun, J.A. McCammon, Mathematical and numerical aspects of the adaptive fast multipole Poisson-Boltzmann solver, Commun. Comput. Phys. 13 (2013) 107–128.

[35] Z.H. Qiao, Z.L. Li, T. Tang, A finite difference scheme for solving the nonlinear Poisson-Boltzmann equation modeling charged spheres, J. Comput. Math. 24 (2006) 252–264.

[36] S.N. Yu, W.H. Geng, G.W. Wei, Treatment of geometric singularities in implicit solvent models, J. Chem. Phys. 126 (2007) 244108.

[37] Y.C. Zhou, M. Feig, G.W. Wei, Highly accurate biomolecular electrostatics in continuum dielectric environments, J. Comput. Chem. 29 (2008) 87–97.

[38] W.H. Geng, G.W. Wei, Multiscale molecular dynamics using the matched interface and boundary method, J. Comput. Phys. 230 (2011) 435–457.

[39] W.H. Geng, S.N. Yu, G.W. Wei, Treatment of charge singularities in implicit solvent models, J. Chem. Phys. 127 (2007) 114106.

[40] Q. Cai, J. Wang, H.-K. Zhao, R. Luo, On removal of charge singularity in Poisson-Boltzmann equation, J. Chem. Phys. 130 (2009) 145101.

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

[42] H. Cheng, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, J. Comput. Phys. 155 (1999) 468–498.

[43] L. Greengard, J.F. Huang, A new version of the fast multipole method for screened Coulomb interactions in three dimensions, J. Comput. Phys. 180 (2002) 642–658.

[44] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–859.

[45] J.P. Bardhan, M.D. Altman, D.J. Willis, S.M. Lippow, B. Tidor, J.K. White, Numerical integration techniques for curved-element discretizations of molecule-solvent interfaces, J. Chem. Phys. 127 (2007) 014701.

[46] M.A. Golberg, C.S. Chen, *Discrete Projection Methods for Integral Equations* (Computational Mechanics Publications, Southampton, U.K., 1997)

[47] J. Barnes, P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, Nature 324 (1986) 446-449.

[48] Z.-H. Duan, R. Krasny, An adaptive treecode for computing nonbonded potential energy in classical molecular systems, J. Comput. Chem. 22 (2001) 184–195.

[49] K. Lindsay, R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, J. Comput. Phys. 172 (2001) 879–907.

[50] P. Li, H. Johnston, R. Krasny, A Cartesian treecode for screened Coulomb interactions, J. Comput. Phys. 228 (2009) 3858–3868.

[51] M.F. Sanner, A.J. Olson, J.C. Spehner, Reduced surface: An efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.

[52] MSMS website, mgl.scripps.edu/people/sanner/html/msms_home.html

[53] Netlib repository, http://www.netlib.org/

[54] J.G. Kirkwood, Theory of solution of molecules containing widely separated charges with special application to Zwitterions, J. Chem. Phys. 7 (1934) 351–361.

[55] D.M. Briercheck, T.C. Wood, T.J. Allison, J.P. Richardson, G.S. Rule, The NMR structure of the RNA binding domain of *E. coli* rho factor suggests possible RNA-protein interactions, Nature Struct. Biol. 5 (1998) 393–399.

[56] Protein Data Bank, www.rcsb.org/pdb/home/home.do

[57] A.D. MacKerell Jr., D. Bashford, M. Bellott, J.D. Dunbrack, M.J. Evanseck, M.J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F.T.K. Lau, C. Mattos, S. Michnick, T. Ngo, D.T. Nguyen, B. Prodhom, W.E. Reiher, B. Roux, M. Schlenkrich, J.C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, M. Karplus, All-atom empirical potential for molecular modeling and dynamics studies of proteins, J. Phys. Chem. B 102 (1998) 3586–3616.

[58] APBS website, www.poissonboltzmann.org/apbs

[59] W. Humphrey, A. Dalke, K. Schulten, VMD–Visual Molecular Dynamics, J. Mol. Graph. 14 (1996) 33–38.

[60] M.S. Warren, J.K. Salmon, A portable parallel particle program, Comput. Phys. Commun. 87 (1995) 266–290.

[61] A. Grama, V. Kumar, A. Sameh, Scalable parallel formulations of the Barnes-Hut method for n-body simulations, Parallel Computing 24 (1998) 797-822.

[62] Y.M. Marzouk, A.F. Ghoniem, K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical N-body simulations, J. Comput. Phys. 207 (2005) 493-528.

[63] M. Winkel, R. Speck, H. Hübner, L. Arnold, R. Krause, P. Gibbon, A massively parallel, multi-disciplinary Barnes-Hut tree code for extreme-scale N-body simulations, Comput. Phys. Commun. 183 (2012) 880–889.

[64] W. Smith, Molecular dynamics on hypercube parallel computers, Comput. Phys. Commun. 62 (1991) 229–248.

[65] D. Liu, Z.-H. Duan, R. Krasny, J. Zhu, Parallel implementation of the treecode Ewald method, *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IEEE Computer Society Press, Santa Fe, New Mexico, 2004)

[66] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface* (The MIT Press, Cambridge, Massachusetts, 1994)

[67] W. Geng, Parallel higher-order boundary integral electrostatics computation on molecular surfaces with curved triangulation, J. Comput. Phys. 241 (2013) 253–265.

[68] P.W. Bates, G.W. Wei, S. Zhao, Minimal molecular surfaces and their applications, J. Comput. Chem. 29 (2008) 380–391.

[69] Z. Yu, M.J. Holst, Y. Cheng, J.A. McCammon, Feature-preserving adaptive mesh generation for molecular shape modeling and simulation, J. Mol. Graph. Model. 26 (2008) 1370–1380.

[70] C.L. Bajaj, G. Xu, Q. Zhang, A fast variational method for the construction of resolution adaptive C^2-smooth molecular surfaces, Comput. Methods Appl. Mech. Engrg. 198 (2009) 1684–1690.

[71] D. Xu, Y. Zhang, Generating triangulated macromolecular surfaces by Euclidean Distance Transform, PLoS ONE 4(12): e8140. doi:10.1371/journal.pone.0008140

[72] M.X. Chen, B.Z. Lu, TMSmesh: A robust method for molecular surface mesh generation using a trace technique, J. Chem. Theory Comput. 7 (2011) 203–212.

[73] C.J. Cramer, D.G. Truhlar, Implicit solvation models: Equilibria, structure, spectra, and dynamics. Chem. Rev. 99 (1999) 2161–2200.

[74] D.M. Chipman, Solution of the linearized Poisson-Boltzmann equation, J. Chem. Phys. 120 (2004) 5566–5575.

[75] J. Tomasi, B. Mennucci, R. Cammi, Quantum mechanical continuum solvation models, Chem. Rev. 105 (2005) 2999–3093.

[76] A.W. Lange, J.M. Herbert, A simple polarizable continuum solvation model for electrolyte solutions, J. Chem. Phys. 134 (2011) 204110.