Journal of Computational Physics 373 (2018) 750–762

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

On preconditioning the treecode-accelerated boundary integral (TABI) Poisson–Boltzmann solver

Jiahui Chen, Weihua Geng *

Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

ARTICLE INFO

Article history:
Received 10 March 2018
Received in revised form 4 July 2018
Accepted 6 July 2018
Available online 17 July 2018

Keywords:
Treecode
Electrostatic
Boundary integral
Poisson–Boltzmann
Preconditioning
GMRES

ABSTRACT

We recently developed a treecode-accelerated boundary integral (TABI) solver for solving Poisson–Boltzmann (PB) equation [1]. The solver has combined advantages in accuracy, efficiency, memory, and parallelization as it applies a well-posed boundary integral formulation to circumvent many numerical difficulties associated with the PB equation and uses an $O(N \log N)$ treecode to accelerate the GMRES iterative solver. However, as observed in our previous work [2], occasionally when the mesh generator produces low quality triangles, the number of GMRES iterations required to solve the discretized boundary integral equations $Ax = b$ could be large. To address this issue, we design a preconditioning scheme using preconditioner matrix M such that $M^{-1}A$ has much improved condition while $M^{-1}z$ can be rapidly computed for any vector z. In this scheme, the matrix M carries the interactions between boundary elements on the same leaf only in the tree structure thus is block diagonal with many computational advantages. The sizes of the blocks in M are conveniently controlled by the treecode parameter N_0, the maximum number of particles per leaf. The numerical results show that this new preconditioning scheme improves the TABI solver with significantly reduced iteration numbers and better accuracy, particularly for protein sets on which TABI solver previously converges slowly. In addition, this preconditioning scheme potentially can improve the condition number of various multipole method accelerated boundary elements solvers in scattering, fluids, elasticity, etc.

Published by Elsevier Inc.

1. Introduction

In biomolecular simulations, electrostatic interactions are of paramount importance due to their ubiquitous existence and significant contribution in the entire force fields. However, computing these nonbonded interactions is challenging since they are pairwise at cost of $O(N^2)$ and long range [3]. To reduce the degree of freedom of the system in terms of electrostatic interactions, implicit solvent Poisson–Boltzmann (PB) model is used [4], in which the water molecules are treated as continuum and the dissolved electrolytes are approximated using the statistical Boltzmann distribution. The PB model has broad application in biomolecular simulations such as protein structure [5], protein–protein interaction [6,7], chromatin packing [8], pKa [9–12], membrane [13,14], binding energy [15–17], solvation free energy [18,19], ion channel profiling [20], etc.

* Corresponding author.
E-mail address: wgeng@smu.edu (W. Geng).

https://doi.org/10.1016/j.jcp.2018.07.011
0021-9991/Published by Elsevier Inc.

The PB equation is an elliptic interface problem with several numerical difficulties such as discontinuous dielectric coefficients, singular source, complex interface, and infinity boundary condition. Standard finite difference discretization in solving PB equation is efficient and robust thus popular [21–24], however it may suffer from accuracy reduction due to discontinuity of the coefficients, non-smoothness of the solution, singularity of the sources, and truncation of the domains, unless special interface and singularity treatments are applied [25,26] at the price of more complicated discretization scheme and possibly reduced convergence speed in iteration. Meanwhile, boundary integral methods are effective alternatives, which analytically circumvent above-mentioned difficulties. In addition, due to the structures hidden in the linear algebraic system after the discretization of the boundary integral and molecular surface, the matrix-vector product in each iteration can be accelerated by fast methods such as fast multipole methods (FMM) and treecode [27,28]. Our recently developed treecode-accelerated boundary integral (TABI) Poisson–Boltzmann solver is such an example [1] combining the advantages of both boundary integral equation and multipole methods. The TABI solver uses the well-posed derivative form of the Fredholm second kind integral equation [29] and the $O(N \log N)$ treecode [28] combined to solve the PB equation efficiently and accurately. It also has advantage in memory use and parallelization [1,30]. The TABI solver has been used by many computational biophysics/biochemistry groups and it has been disseminated standalone or as a contributive module of the popular APBS software package [31,32].

A bottleneck that hinders the efficiency of the TABI solver, which only uses the simplest diagonal or Jacobi preconditioning is at the mesh quality for triangulating the large and complex molecular surfaces. Our numerical tests previously showed that although the adopted integral formulation is well-posed [29], the mesh quality for triangulating the complex molecular surface affects the convergence speed of GMRES [1,2]. Currently our choice of the triangular mesh generator is the MSMS package developed by Sanner et al. [33], which is very efficient in generating triangular meshes for given biomolecules. However, due to the complexity of the molecular surface, the produced triangles could be irregularly shaped e.g. small in size, large in angle ($\approx \pi$), or in some other shapes which might affect the iterative convergence but cannot be filtered by our preprocessing subroutines. To resolve these issues, on one hand we are seeking better choices for molecular surface triangulation, and on the other hand we are trying to find solution to reduce the effect of mesh quality.

In the present work, we provide a newly designed preconditioning scheme, which cancels the slow-down effects caused by the mesh quality, while the added computational cost due to preconditioning is negligibly small. Our numerical simulation shows that for many tested proteins on which the TABI solver used to converge slowly now converges rapidly with this update. In addition, we believe this preconditioning scheme can benefit many multipole methods accelerated boundary integral Poisson–Boltzmann solvers such as [34–44]. The similar ideas can also be used to accelerate solving boundary integral equations from other areas such as scattering, fluids, elasticity, etc.

We next provide theories and algorithms related to the TABI solver and its preconditioning, followed by numerical results and discussion. This paper ends with a concluding remark section.

2. Theory and algorithms

In this section, we briefly describe the Poisson–Boltzmann (PB) implicit solvent model, review the current PB solvers, and introduce our recently developed treecode-accelerated boundary integral (TABI) PB solver, followed by our preconditioning scheme.

2.1. The Poisson–Boltzmann (PB) model for a solvated biomolecule

The PB model for a solvated biomolecule is depicted in Fig. 1(a) in which the molecular surface Γ separates the solute domain Ω_1 from the solvent domain Ω_2. Fig. 1(b) is an example of the molecular surface Γ as the triangulated surface of protein barnase [6]. In domain Ω_1, the solute is represented by N_c partial charges q_k located at atomic centers $\mathbf{r}_k$ for $k = 1, \cdots, N_c$, while in domain Ω_2, a distribution of ions is described by a Boltzmann distribution and we consider a linearized version in this study. The solute domain has a low dielectric constant ϵ_1 and the solvent domain has a high dielectric constant ϵ_2. The modified inverse Debye length $\bar{\kappa}$ is given as $\bar{\kappa}^2 = \epsilon_2 \kappa^2$, where κ is the inverse Debye length measuring the ionic strength; $\bar{\kappa} = 0$ in Ω_1 and is nonzero only in Ω_2. The electrostatic potential $\phi(\mathbf{x})$ satisfies the linear PB equation,

$$-\nabla \cdot \epsilon(\mathbf{x}) \nabla \phi(\mathbf{x}) + \bar{\kappa}^2(\mathbf{x}) \phi(\mathbf{x}) = \sum_{k=1}^{N_c} q_k \delta(\mathbf{x} - \mathbf{x}_k), \tag{1}$$

subject to continuity conditions for the potential and electric flux density on Γ,

$$[\phi] = 0, \quad [\epsilon \phi_\nu] = 0, \tag{2}$$

where $[f] = f_1 - f_2$ is the difference of the quantity f across the interface, and $\phi_\nu = \partial \phi / \partial \nu$ is the partial derivative in the outward normal direction ν. The model also incorporates the far-field boundary condition,

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

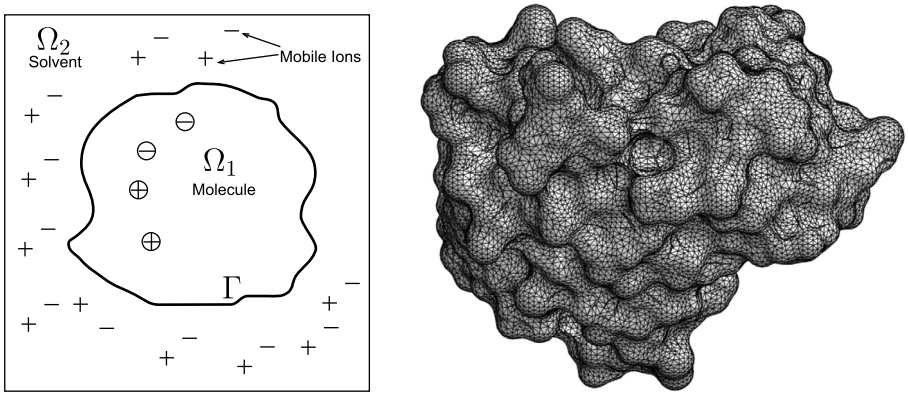

Fig. 1. Schematic models; (a) the PB implicit solvent model, in which the molecular surface Γ separates space into the solute region Ω_1 and solvent region Ω_2; (b) the triangulation of molecular surface of protein Barnase at MSMS density $d = 5$ (# of vertices per Å^2).

Note that Eqs. (1)–(3) define a boundary value problem for the potential $\phi(\mathbf{x})$ which in general must be solved numerically.

2.2. Numerical PB solvers

Numerical methods for the PB model face several challenges: the solute is represented by singular point charges, the molecular surface is often geometrically complex, the dielectric function is discontinuous across the surface, and the domain is unbounded. Two types of methods have been developed, (1) grid-based finite-difference and finite-element methods that discretize the entire volumetric domain, e.g. [23,24,45–51], and (2) boundary element methods that discretize the molecular surface, e.g. [1,29,34–40,42–44,52]. The reader may consult [4,53] for comprehensive reviews of numerical PB solvers.

Grid-based PB solvers are widely used and available in many software packages, e.g. APBS [21], AMBER [22,47], CHARMM [23], Delphi [24,54]. In these schemes, the singular charges are interpolated to the grid or regularized using Green's function, the interface conditions are captured approximately, and the far-field boundary condition is enforced on a truncated domain. While these errors can be reduced by refining the grid and enlarging the truncated domain, schemes with higher order accuracy have been developed to enforce the interface conditions more strictly [16,25,55–58]. Some of these methods developed by mathematician are disseminated toward the greater bioscience community in the forms of web-servers e.g. MIBPB (http://weilab.math.msu.edu/MIBPB/) for the finite difference solver [25] and SMPBS (http://smpbs.math.uwm.edu) for the finite element solver [57].

Our approach to these issues is to employ a boundary element method (BEM) in which the boundary integral PB equation is solved and the singular charges, interface conditions, and far-field boundary condition are treated analytically. The resulting BEM solvers are advantageous in that the molecular surface is represented more accurately and they avoid the expense of a volumetric grid. In a conventional BEM, these advantages are offset by the cost of evaluating the $O(N^2)$ interactions among the N elements representing the molecular surface. However, fast summation schemes are now available to reduce the cost, such as the Fast Multipole Method (FMM) [27,36,37,40,59] and the treecode [28,60]. Our choice for solving the PB equations is a BEM accelerated by a Cartesian treecode [1], which is described in the next section.

2.3. Treecode-accelerated boundary integral (TABI) PB solver

In this section we describe our recently developed TABI PB solver for computing the electrostatic surface potential and solvation energy [1]. We present the boundary integral form of the PB implicit solvent model, the discretization of the boundary integral equations, and the treecode algorithm for accelerating the matrix-vector product.

2.3.1. Boundary integral form of PB model

This section summarizes the well-conditioned boundary integral form of the PB implicit solvent model we employ [1,29]. Applying Green's second identity and properties of fundamental solutions to Eq. (1) yields the electrostatic potential in each domain,

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

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

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

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

Then applying the interface conditions in Eq. (2) with the differentiation of electrostatic potential in each domain yield a set of boundary integral equations relating the surface potential ϕ_1 and its normal derivative $\partial\phi_1/\partial\nu$ on Γ,

$$\frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}) = \int_\Gamma \left[K_1(\mathbf{x},\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu} + K_2(\mathbf{x},\mathbf{y})\phi_1(\mathbf{y})\right] dS_\mathbf{y} + S_1(\mathbf{x}), \quad \mathbf{x}\in\Gamma, \tag{6a}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial\phi_1(\mathbf{x})}{\partial\nu} = \int_\Gamma \left[K_3(\mathbf{x},\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu} + K_4(\mathbf{x},\mathbf{y})\phi_1(\mathbf{y})\right] dS_\mathbf{y} + S_2(\mathbf{x}), \quad \mathbf{x}\in\Gamma, \tag{6b}$$

where $\varepsilon = \varepsilon_2/\varepsilon_1$. As given in Eqs. (7a)–(7b) and (10), the kernels $K_{1,2,3,4}$ and source terms $S_{1,2}$ are linear combinations of G_0, G_k, and their first and second order normal derivatives [1,29].

$$K_1(\mathbf{x},\mathbf{y}) = G_0(\mathbf{x},\mathbf{y}) - G_\kappa(\mathbf{x},\mathbf{y}), \quad K_2(\mathbf{x},\mathbf{y}) = \varepsilon\frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{y}} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{y}}, \tag{7a}$$

$$K_3(\mathbf{x},\mathbf{y}) = \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{x}} - \frac{1}{\varepsilon}\frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{x}}, \quad K_4(\mathbf{x},\mathbf{y}) = \frac{\partial^2 G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{x}\partial\nu_\mathbf{y}} - \frac{\partial^2 G_0(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{x}\partial\nu_\mathbf{y}}, \tag{7b}$$

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

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial\nu_\mathbf{x}} = -\nu(\mathbf{x})\cdot\nabla_\mathbf{x} G(\mathbf{x},\mathbf{y}) = -\sum_{m=1}^{3}\nu_m(\mathbf{x})\partial_{x_m} G(\mathbf{x},\mathbf{y}), \tag{8}$$

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

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

and the source terms $S_{1,2}$ are

$$S_1(\mathbf{x}) = \frac{1}{\varepsilon_1}\sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad S_2(\mathbf{x}) = \frac{1}{\varepsilon_1}\sum_{k=1}^{N_c} q_k\frac{\partial G_0(\mathbf{x},\mathbf{y}_k)}{\partial\nu_\mathbf{x}}. \tag{10}$$

Once the potential and normal derivative of the potential from Eqs. (6a)–(6b) are solved, potential at any point in the space can be computed via Eqs. (4a)–(4b) or a numerically more accurate formulation as mentioned in [29]. The electrostatic solvation energy can also be obtained by

$$E_{\text{sol}} = \frac{1}{2}\sum_{k=1}^{N_c} q_k\phi_{\text{reac}}(\mathbf{y}_k) = \frac{1}{2}\sum_{k=1}^{N_c} q_k\int_\Gamma\left[K_1(\mathbf{y}_k,\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu} + K_2(\mathbf{y}_k,\mathbf{y})\phi_1(\mathbf{y})\right] dS_\mathbf{y}, \tag{11}$$

where $\phi_{\text{reac}}(\mathbf{x}) = \phi(\mathbf{x}) - S_1(\mathbf{x})$ is the reaction potential [1,29]. In our numerical results, we report many results involving solving PB equation and calculating the electrostatic solvation energy.

2.3.2. Discretization of boundary integral equations

The molecular surface is triangulated using MSMS [33]. For example, Fig. 1 (b) shows the triangulated molecule surfaces at MSMS density $d = 5$ (# of vertices per Å^2) of protein barnase, which will bind another protein barstar to form a biomolecular complex (PDB: 1b2s) [6]. The integrals in Eqs. (6a)–(6b) are discretized by centroid collocation. Letting $\mathbf{x}_i, i = 1,\cdots,N$ denote the triangle centroids of the N triangular elements, the discretized Eqs. (6a)–(6b) have the following form for $i = 1,\ldots,N$,

$$\frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}_i) = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_1(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu} + K_2(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right]\Delta s_j + S_1(\mathbf{x}_i), \tag{12a}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial\phi_1(\mathbf{x}_i)}{\partial\nu} = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_3(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu} + K_4(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right]\Delta s_j + S_2(\mathbf{x}_i), \tag{12b}$$

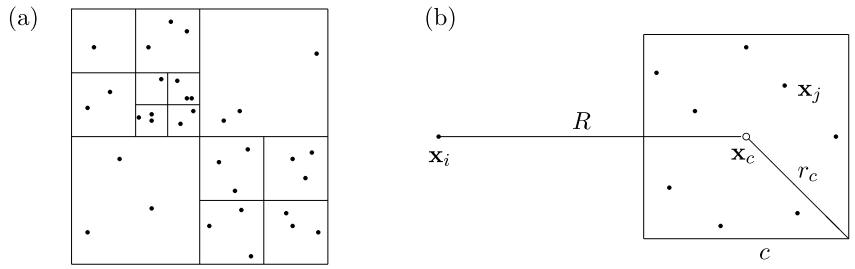

Fig. 2. Details of treecode; (a) tree structure of particle clusters; (b) particle–cluster interaction between particle $\mathbf{x}_i$ and cluster $c = \{\mathbf{x}_j\}$; $\mathbf{x}_c$: cluster center, R: particle–cluster distance, r_c: cluster radius.

where Δs_j is the area of the jth boundary element for $j = 1, \cdots, N$. The term $j = i$ is omitted to avoid the kernel singularity. Equations (12a)–(12b) is a linear system $Ax = b$, where x contains the surface potentials $\phi_1(\mathbf{x}_i)$ and normal derivatives $\frac{\partial \phi_1(\mathbf{x}_i)}{\partial \nu}$, weighted by the element area Δs_i, and $\mathbf{b}$ contains the source terms $S_1(\mathbf{x}_i)$, $S_2(\mathbf{x}_i)$. The system is solved by the GMRES iteration, which requires a matrix-vector product in each step [61]. Since the matrix is dense, computing the product by direct summation requires $O(N^2)$ operations, which is prohibitively expensive when N is large. In the next section we describe the treecode algorithm used to accelerate the matrix-vector product.

2.3.3. Treecode

We summarize the treecode algorithm and refer to previous work for more details [28,60,62,63]. The matrix-vector product Ax for Eqs. (12a)–(12b) has the form of N-body potentials,

$$V_i = \sum_{j=1, j\neq i}^{N} q_j G(\mathbf{x}_i, \mathbf{x}_j), \quad i = 1, \ldots, N, \tag{13}$$

where G is a kernel, $\mathbf{x}_i, \mathbf{x}_j$ are centroids (also called particles in this context), and q_j is a charge associated with $\mathbf{x}_j$. To this end, the q_j in Eq. (13) is equivalent to the $\Delta s_j \phi_1(\mathbf{x}_j)$ or $\Delta s_j \frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu}$ in Eqs. (12a)–(12b) and G is one of the kernels of K_{1-4}. To evaluate the potentials V_i rapidly, the particles $\mathbf{x}_i$ are divided into a hierarchy of clusters having a tree structure in a 2-D illustration as in Fig. 2(a). The root cluster is a cube containing all the particles and subsequent levels are obtained by dividing a parent cluster into children [60]. The process continues until a cluster has fewer than N_0 particles (N_0 is a user-specified parameter representing the maximum number of particles per leaf, and $N_0 = 3$ in Fig. 2(a)). Then V_i is evaluated as a sum of particle–particle interactions and particle–cluster interactions, which are depicted in Fig. 2(b),

$$V_i \approx \sum_{c \in N_i} \sum_{\mathbf{x}_j \in c} q_j G(\mathbf{x}_i, \mathbf{x}_j) + \sum_{c \in F_i} \sum_{||\mathbf{k}||=0}^{p} a^{\mathbf{k}}(\mathbf{x}_i, \mathbf{x}_c)\, m_c^{\mathbf{k}}, \tag{14}$$

where c denotes a cluster, and N_i, F_i denote the near-field and far-field clusters 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 $\mathbf{x}_i$ [28]. 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 multipole acceptance criterion (MAC) is satisfied, $r_c / R \leq \theta$, where r_c is the cluster radius, $R = |\mathbf{x}_i - \mathbf{x}_c|$ is the particle–cluster distance, and θ is a user-specified parameter [60].

The accuracy and efficiency of the treecode is controlled by the combination of order p, MAC parameter θ, and maximum particles per leaf N_0. Using the treecode, the operation count for the matrix-vector product is $O(N \log N)$; the factor N is the number of particles $\mathbf{x}_i$, and the factor $\log N$ is the number of levels in the tree.

2.4. Preconditioning

In order to precondition Krylov subspace methods in solving $Ax = b$, taking left-precondition scheme as an example, we seek a preconditioning matrix or a preconditioner M such that two conditions are satisfied:

(1) M is similar to A such that $M^{-1}A$ has improved condition compared to A thus less number of iterations are required in solving $M^{-1}Ax = M^{-1}b$ compared to solving $Ax = b$;
(2) $M^{-1}z = y$ can be efficiently computed, which is equivalent to solving y from $My = z$.

Conditions (1) and (2) cannot be improved concurrently and a tradeoff must be made. For examples, $M = A$ is the extreme of condition (1) and $M = I$ the identity matrix is the extreme of condition (2), and the preconditioner we are seeking lies in between these two extremes.

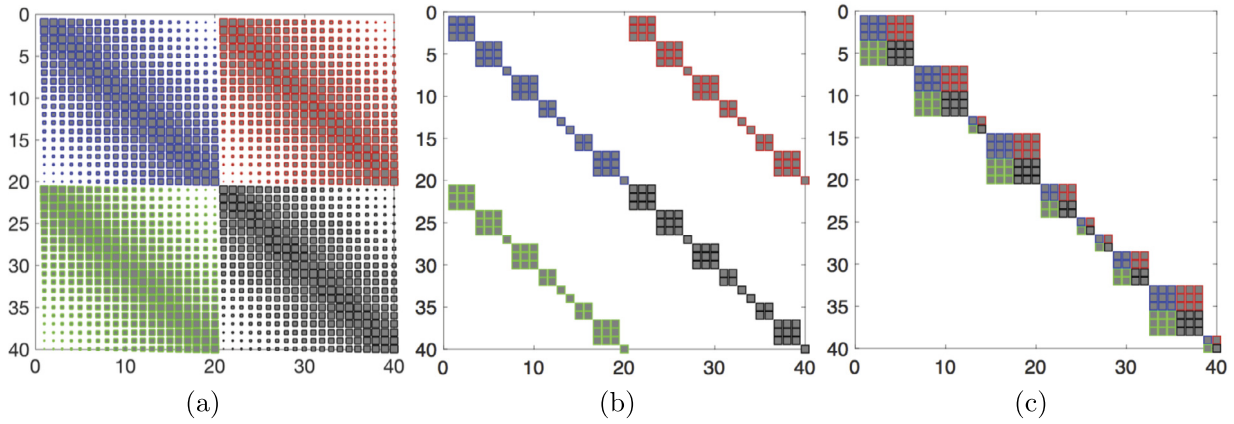

Fig. 3. A schematic illustration of the boundary element dense matrix A and its preconditioning matrix M: (a) matrix A for the case of $N = 20$ elements (the size of the matrix entry shows the strength of the interaction; the four different color-coded region relates to K_{1-4} in Eqs. (12a)–(12b)); (b) the "block diagonal block" preconditioning matrix M ($N_0 = 3$ in this schematic illustration and there are 10 leaves with 1-3 particles/elements each); (c) the "block diagonal" preconditioning matrix M, which is a permuted matrix from M in (b) after switching the order of the unknowns. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The design of our preconditioner is motivated from the observation that in electrostatic interactions, which is also the interactions between boundary elements in solving integral equations, the short range interactions are smaller in number of interactions but more significant in strength than the long range interactions, which are large in number of interactions and computationally more expensive. Due to their large numbers, the long interactions are calculated by multipole expansions. This gives us the ideas that for a preconditioner of A, we might use short range interactions represented by the matrix M to approximate all interactions, ignoring long range interactions. We have three ideas.

(1) interactions between elements computed in direct summation, which is the first term in Eq. (14), or simply,
(2) interactions between elements on the same leaf only (i.e. there is no interaction between elements from different leaves, even these elements are close from each other), which is smaller subsets of the elements in idea (1), or more complicated,
(3) interactions between elements geometrically close at a specified level of the tree, which is a bigger set containing elements in idea (1).

Among these ideas, idea (2) has the least computational cost but the weakest preconditioning power and idea (3) has the most computational cost but the strongest preconditioning power. As a neutral choice, we start with idea (1) and we immediately encounter a difficulty. We are excited to see that solving $M^{-1}Ax = M^{-1}b$ receives much improved convergence speed but we also realize solving $My = z$, which requires another GMRES solver, suffers from similar slow convergence if solving $Ax = b$ shows slow convergence. Here M is a much simplified version of A with only the near-field direct sum interactions. This difficulty implies that it should be the very near interactions between problematic triangular elements cause the slow GMRES convergence. Since $M^{-1}A$ has much improved condition, the key now is to find a kind of M such that $My = z$ can be solved rapidly.

It turns out that idea (2), which is our final choice, has great advantages in efficiency and accuracy for solving $My = z$. To be more precise, we give the explicit definition of A and M as from the discretized system (12a)–(12b). Let $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \in \mathbb{R}^{2N\times 2N}$, where $A_{mn} \in \mathbb{R}^{N\times N}$ for $m, n = 1, 2$. Then the entries of these block matrices are given as

$$\begin{aligned} &A_{11}(i,j) = \frac{1}{2}(1+\varepsilon)\,\delta_{ij} + K_2(\mathbf{x}_i,\mathbf{x}_j)\Delta s_j(1-\delta_{ij}), \quad A_{12}(i,j) = K_1(\mathbf{x}_i,\mathbf{x}_j)\Delta s_j(1-\delta_{ij}), \\ &A_{21}(i,j) = K_4(\mathbf{x}_i,\mathbf{x}_j)\Delta s_j(1-\delta_{ij}), \qquad\qquad A_{22}(i,j) = \frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\delta_{ij} + K_3(\mathbf{x}_i,\mathbf{x}_j)\Delta s_j(1-\delta_{ij}), \end{aligned} \tag{15}$$

for $i, j = 1, \cdots, N$, where $\delta_{i,j}$ is the Kronecker delta function and Δs_j is the area of the jth element. The definition of M will be essentially similar to A except that the entries of M are zero if i and j are not on the same leaf of the tree, i.e.

$$M_{m,n}(i,j) = \begin{cases} A_{m,n}(i,j) & \text{if } i, j \text{ are on the same leaf} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

for $i, j = 1, \cdots, N$ and $m, n = 1, 2$.

Here we use Fig. 3 to illustrate the design and advantage of our preconditioning scheme. Fig. 3(a) is the illustration of the dense boundary element matrix A for the discretized system (12a)–(12b) with 20 boundary elements. The four

different colors represent the four kernels K_{1-4} related entries of the linear algebraic matrix A in Eqs. (12a)–(12b). Note the unknowns are ordered by the potentials ϕ_1 on all elements, followed by the normal derivative of the potential $\frac{\partial \phi_1}{\partial \nu}$. The size of the matrix entry in Fig. 3 indicates the magnitude of the interaction between a target element and a source element, which decays from the main diagonal to its two wings. By only including the interactions between elements in the same leaf, we obtain our designed preconditioning matrix M as illustrated in Fig. 3(b). This preconditioning matrix M has four blocks, and each block is a diagonal block matrix. Rigorously, we should call M a "block diagonal block matrix", but we use the term "block diagonal matrix" for simplification and the reason explained next.

The most important advantage of this designed preconditioning matrix M is that computing $y = M^{-1}z$ or equivalently solving y from $My = z$ can be rapidly computed using direct methods e.g. LU factorization. To see this, we rearrange the unknowns in vector y. Originally, y is composed of the first segment of potential on all elements followed the second segment of normal derivative of the potential on all elements. The rearranged y has N_l segments, where N_l is the total number of leaves in the tree structure. One leaf after the other, each leaf segment of the rearranged y consists of the potentials on elements belonging to the leaf followed by normal derivatives of the potential on elements of the same leaf. This rearrangement creates a block diagonal matrix M as illustrated in Fig. 3(c). This rearranged matrix M is equivalent to the original matrix M in the sense of condition number, eigenvalues, singular values but is much more efficient and convenient at solving $My = z$ and computing characteristic quantities of M. For simplicity and the equivalence mentioned here, we do not distinct the original M and the rearranged M in the context.

Since $M = \text{diag}\{M_1, M_2, \cdots, M_{N_l}\}$ as shown in Fig. 3(c) is a block diagonal matrix such that $My = z$ can be solved using direct method e.g. LU factorization by solving each individual $M_i y_i = z_i$. Here each M_i is a square nonsingular matrix, which represents the interaction between particles/elements on the ith leaf of the tree. It worths noting that the efficiency is not affected even when M_i has a large condition number since direct solver is used for solving $My = z$. Meanwhile, we can compute the condition number, eigenvalues, singular values of M easily. These values of M are similar or close to corresponding values of matrix A thus are very useful to study the properties of A. For example, the norm-2 condition number of M can be obtained by its singular values as $\text{cond}_2(M) = \sigma_{\max}/\sigma_{\min}$, where the singular values of M can be obtained from the singular values of each sub-matrix M_i.

We also provide the computational cost estimate for the preconditioning scheme, which is essentially the cost of solving $My = z$. Let N_0 be the number of particle per leaf, a user specified treecode parameter as explained in previous section. Since we only consider the interactions between particles on the same leaf for preconditioning, the dimension of the matrix M_i is less than or equal to N_0 and we use the upper limit N_0 to represent it. The total number of blocks is N/N_0. The total cost of solving $My = z$ by solving all $M_i y_i = z_i$ is therefore $O(N_0^3 N/N_0) = O(NN_0^2)$, which is essentially $O(N)$ if small N_0 is used. Meanwhile, N_0 cannot be too small or M will not include the singular element interactions to cancel with the corresponding interaction in A. These considerations suggest we use small N_0 e.g. 100 in our preconditioning scheme, which is justified by the numerical results in the next section.

3. Results

In this section, we provide numerical results using block diagonal preconditioning as compared with the previously used diagonal preconditioning in TABI solver [1]. Our numerical results are produced on a desktop with i5 7500 CPU and 8G Memory, using GNU Fortran 7.2.0 compiler with compiling option "-O3". We fix some PB model parameters (relative dielectric constants $\epsilon_1 = 1$ and $\epsilon_2 = 80$, inverse Debye length $\kappa = 0.1257/$ Å^2, which corresponds to ionic strength 0.15 molar) and some treecode parameters ($p = 3$), while modify the more relevant parameters (MSMS density d, the MAC criterion θ, the maximum particle per leaf N_0) as described below. All protein structures are obtained from Protein Data Bank (https://www.wwpdb.org) and partial charges are assigned by CHARMM22 force field [64] using PDB2PQR software [65]. For dissemination to the greater science community, all codes are published on sourceforge (https://sourceforge.net/projects/tabipb/) following the New BSD License. The user can also use the link from the corresponding author's website to access the code. In the following, we first solve the PB equation and compute electrostatic solvation energy on two selected proteins using the TABI solver with block diagonal preconditioning scheme at various parameters to justify an optimal choice of these parameters. Following that, we show the improvements of the block diagonal preconditioning scheme compared with the original diagonal preconditioning scheme on selected proteins.

The physical quantity we computed in this manuscript is the electrostatic free energy of solvation with the unit kcal/mol. The electrostatic potential ϕ or ϕ_1 governed in Eq. (1) or Eqs. (6a)–(6b) uses the unit of $e_c/(4\pi$ Å), where e_c is the elementary charge. By doing this, we can directly use the partial charge obtained from PDB2PQR [65] for solving the PB equation. After obtaining the potential, we can convert the unit $e_c/(4\pi$ Å) to kcal/mol/e_c by multiplying the constant $4\pi 332.0716$ at room temperature $T = 300$ K. From potential to energy, only a multiplication of e_c is needed. More details about PB equation related units conversion can be found in [57,66].

3.1. The selection of treecode parameters

The treecode used by the TABI solver has three tree-structure related parameters: the order p of Taylor expansion, which determines the truncation error of the treecode; the MAC criterion θ, which majorly determines the fraction of using treecode (particle–cluster) or direct sum (particle–particle) for electrostatic interactions; and the maximum particle per

Table 1
Treecode parameter selection (θ and N_0) for preconditioned TABI solver: MSMS density $d = 10$, protein 1bpi has 60600 elements and protein 1a7m has 147121 elements; CPU time t in seconds are reported in terms of total time, time for computing one Ax in each iteration, and time for solving one $My = z$ for the preconditioning matrix M.

θ	N_0	1bpi				1a7m			
		# of it.	t (total)	t (Ax)	t ($My = z$)	# of it.	t (total)	t (Ax)	t ($My = z$)
0.8	**100**	10	**68.1**	6.3	0.4	12	**230.7**	17.9	0.9
	300	9	82.4	6.2	2.4	11	277.2	17.7	6.2
	500	8	124.8	6.4	7.3	10	337.9	18.0	12.8
0.5	100	10	149.2	14.4	0.4	12	518.3	41.9	0.9
	300	9	157.3	14.5	2.4	11	543.2	42.0	6.2
	500	8	192.4	14.8	7.3	10	583.0	42.6	12.7
0.2	100	10	545.8	54.2	0.4	12	2308.9	190.7	0.9
	300	9	507.2	53.5	2.4	11	2157.3	188.7	6.2
	500	8	530.4	57.2	7.3	10	2110.8	195.6	12.7

leaf N_0, which determines the level of the tree. From previous research, we found N_0 is less significant than the other two parameters in affecting the treecode efficiency and accuracy [1,28,30]. However, in present research, we find that N_0 plays a significant role in preconditioning. The effects of these parameters for accuracy, efficiency, and parallelization are studied in detail in our previous work [1,28,30], and we here mainly focus on their effects on preconditioning.

Table 1 reports number of iterations and CPU time for solving boundary integral PB equation on the molecular surface of proteins 1a7m and 1bpi. Protein 1bpi has 898 atoms while protein 1a7m has 2809 atoms. We choose these two proteins because solving PB equation using the GMRES solver with only diagonal preconditioning on them requires large number of iterations [2]. The parameters are listed in the caption of the table. From data associated with these two proteins, we have the following two conclusions:

(1) The MAC θ overally determines the efficiency of the algorithms by controlling how much particle–cluster interactions are used to replace particle–particle interaction. To this end, larger θ e.g. 0.8 is preferred than smaller θ e.g. 0.5, 0.2 for better efficiency at the price of moderately more loss of accuracy. From this table, we can see for both cases, larger θ requires less total CPU time.
(2) The impact of N_0 in preconditioning are two-fold: first smaller N_0 requires less CPU in solving $My = z$ for preconditioning as seen in the "t ($My = z$)" column; second, smaller N_0 means more levels of trees thus more particle–cluster computation is used, which leads to less accurate Ax computation, then more number of iterations may be required as seen in the "# of it." column; the overall effect is the combination of these two factors. For example, smaller N_0 uses less CPU time for $\theta = 0.8$ but uses more CPU time for $\theta = 0.2, 0.5$ as seen in the "t (total)" column for both proteins.

By combining all effects, we recommend the combination of $\theta = 0.8$ and $N_0 = 100$ as the *optimal* choice of parameters, which is highlighted in the table. For instance, the diagonal preconditioned TABI solver use 67 iterations for protein 1a7m and 110 iterations for protein 1bpi as reported in [2] while the number of iterations required using block diagonal preconditioning is approximately 10 for both proteins. Note that the underlined number 110 indicates that the maximum number of iterations are reached before the 1.0e−4 GMRES relative error tolerance is satisfied. It is also very clear that when $N_0 = 100$, the time for solving ($My = z$) is much less than the time for computing Ax in each iteration thus cost from block preconditioning is only a small amount of additional cost. We also solve PB equation on these two proteins using even smaller N_0, e.g. $N_0 = 50, 25$. The CPU time using these even smaller N_0 is very close to $N_0 = 100$. However, smaller N_0 might cause increased number of iterations if the singular elements interaction are not included in M matrix to be canceled with that in A as mentioned previously. In the tests shown next, we fix the parameters $\theta = 0.8$ and $N_0 = 100$.

3.2. Preconditioning performance on protein 1a7m at various densities

We next choose the largest protein 1a7m from our test protein sets to check the performance of the preconditioning scheme at various densities, which lead to various numbers of boundary elements.

Table 2 reports electrostatic solvation free energy, number of iterations, and CPU time for solving PB equations using TABI solver on the molecular surface of protein 1a7m at different densities with diagonal preconditioning (d) and block diagonal preconditioning (bd). We increase the density from 1 to 80, resulting in the increased number of elements as seen in the first column of the table. We observe that the solvation energy in columns 2 and 3 are very close, indicating that preconditioning has very limited effects on accuracy. In fact, the values in column 3 could be more accurate for the situation when the GMRES tolerance for using block diagonal preconditioning is well met before the maximum number of iterations (110) is reached, while using the diagonal precondition stops when the maximum number of iterations is reached, without satisfying the desired error tolerance. As an example, it can be seen that at the finest mesh (1306676 elements), using the diagonal preconditioning exits when the maximum number of iterations (110) is reached but the relative residual

Table 2
Convergence performance for protein 1a7m at various densities/# of elements; "d" stands for diagonal preconditioning and "bd" stands for block diagonal preconditioning.

# of ele.	E_{sol} (kcal/mol)		# of its.		CPU time (s)		
	d	bd	d	bd	d	bd	ratio
22491	−2559.64	−2559.64	110	12	208.8	24.9	8.38
48385	−2238.40	−2238.41	110	11	537.7	58.3	9.23
78057	−2213.40	−2214.31	110	28	944.5	253.3	3.73
147121	−2185.65	−2185.63	67	12	1197.9	230.7	5.19
294287	−2176.23	−2176.23	110	13	4357.6	553.1	7.88
620619	−2171.35	−2171.20	110	17	10089.8	1675.9	6.02
1306676	−1995.29	−2168.96	110	30	22964.5	6891.3	3.33

Table 3
Convergence comparison using diagonal preconditioning (d) and block diagonal preconditioning (bd) on a set of 27 proteins; MSMS density $d = 10$.

Index	PDB	# of ele.	E_{sol} (kcal/mol)		# of it.		CPU time (s)			Cond. #
			d	bd	d	bd	d	bd	ratio	M
1	1ajj	40496	−1147.35	−1147.38	9	8	34.8	33.1	1.05	166.3
2	2erl	43214	−963.52	−963.49	9	8	37.2	35.6	1.05	146.3
3	1cbn	44367	−307.39	−307.36	8	7	34.5	33.0	1.05	161.2
4	1vii	47070	−916.82	−916.79	12	10	60.1	53.0	1.13	271.9
5	1fca	47461	−1216.88	−1216.91	9	9	43.6	46.5	0.94	162.1
6	1bbl	49071	−1003.39	−1003.44	10	9	53.4	51.5	1.04	173.9
7	2pde	50518	−826.79	−826.77	**100**	19	517.0	104.7	4.94	**3760.7**
8	1sh1	51186	−761.80	−761.84	**110**	35	587.6	197.6	2.97	**214288.8**
9	1vjw	52536	−1257.74	−1257.79	9	8	46.2	44.5	1.04	154.7
10	1uxc	53602	−1156.69	−1156.68	9	9	51.9	55.1	0.94	151.1
11	1ptq	54256	−884.75	−884.73	10	9	53.0	51.3	1.03	154.6
12	1bor	54628	−865.68	−865.66	11	9	64.0	56.5	1.13	249.4
13	1fxd	54692	−3348.08	−3347.96	8	8	46.6	51.2	0.91	161.3
14	1r69	57646	−1102.50	−1102.43	10	9	62.4	60.6	1.03	185.6
15	1mbg	58473	−1370.88	−1370.90	10	9	64.7	62.7	1.03	409.6
16	1bpi	60600	−1322.44	−1322.43	**110**	10	690.3	67.7	10.19	**89069.7**
17	1hpt	61164	−826.81	−826.82	12	9	79.4	63.9	1.24	231.3
18	451c	79202	−1040.01	−1040.05	19	13	180.4	132.5	1.36	350.4
19	1svr	88198	−1732.81	−1732.82	11	9	112.5	97.7	1.15	425.9
20	1frd	81792	−2894.34	−2894.13	10	10	93.6	100.2	0.93	183.0
21	1a2s	84527	−1942.73	−1942.74	18	13	183.5	141.2	1.30	5816.6
22	1neq	89457	−1757.87	−1757.90	20	10	221.8	117.5	1.89	1170.1
23	1a63	132133	−2407.54	−2407.54	11	10	191.3	185.8	1.03	184.4
24	1a7m	147121	−2185.65	−2185.63	**67**	12	1195.6	229.3	5.21	**3850.6**
25	2go0	111615	−1979.73	−1979.70	20	17	263.8	238.2	1.11	536.1
26	1uv0	128497	−2312.56	−2312.47	**56**	10	873.7	167.3	5.22	**1601.7**
27	4mth	123737	−2501.38	−2501.54	**69**	10	1077.1	165.7	6.50	**4823.5**

error is 9.1e−3, far from the desired tolerance 1.0e−4 thus the solvation energy result is not correct. On the contrary, using block diagonal preconditioning scheme largely reduces number of iterations needed, and returns correct result satisfying the desired tolerance. As in previous test cases, we see much reduced number of iterations in column 5 compared with that from column 4 and much reduced CPU time in column 7 compared with that from column 6, showing the improvement in convergence using block diagonal preconditioning as opposed to using diagonal preconditioning. The similar trend is consistently observed with the refinement of the triangulation.

3.3. Preconditioning performance on a set of 27 proteins

We next provide testing results on a set of 27 proteins, in which the first 24 proteins are from our previous work [2,25] and the last three proteins are HIP/PAP proteins suggested by Dr. Carrie Partch from UC Santa Cruz in studying the pH dependence of the binding and stability [67,68].

Table 3 shows the convergence tests using diagonal preconditioning (d) and block diagonal preconditioning (bd) for a set of 27 proteins. We encountered slow convergence in previous research and now the issue has been well resolved as explained below. The first column of the table is the protein index, followed by PDB ID in the second column, and number of elements in the third column using MSMS density $d = 10$. Columns 4 and 5 are electrostatic solvation energy of the proteins using both preconditioning schemes and we can see the difference is insignificant. We see significant reduction of number of iterations using block diagonal preconditioning (bd) as in column 7 compared with results in column 6 using diagonal preconditioning (d). It worths noting that the worse the diagonal preconditioning result is, the larger improvements

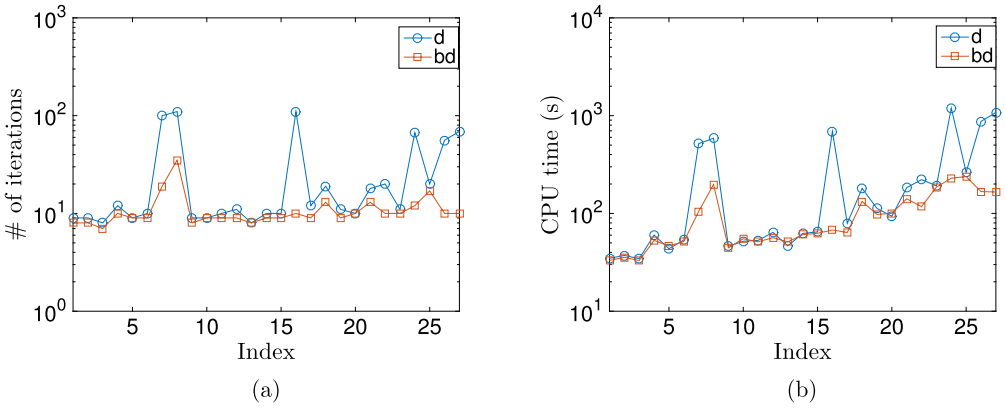

Fig. 4. Convergence comparison using diagonal preconditioning and block diagonal preconditioning. (a) Number of iterations; (b) CPU time (s).

block diagonal preconditioning can achieve. For example, proteins 2pde, 1sh1, 1bpi, 1a7m, 1uv0, and 4mth originally use 100, 110, 110, 67, 56, 60 iterations for diagonal preconditioning but only use 19, 35, 10, 12, 10, 10 iterations for block diagonal preconditioning. The ratio of CPU reduction for these proteins are more than 5 times. The CPU time comparison in columns 8 and 9, as well as its ratio in column 10, further confirms the results we observed in columns 6 and 7 as CPU time is largely related to the number of iterations. We plot the results of columns 6,7,8,9 in Fig. 4, which vividly shows the improvements on both number of iterations and CPU time when block diagonal preconditioning is used to replace the diagonal preconditioning. It shows that although block diagonal preconditioning works particularly better for cases with slow convergence, it does not hurt well-conditioned cases, thus we could uniformly use block diagonal preconditioning to update the original diagonal preconditioning. Figs. 4(a) and 4(b) shows similar pattern as CPU time and number of iterations are highly correlated when GMRES is used to solve the linear algebraic system.

In addition, the last column of Table 3 reports the condition numbers of matrix M for each tested proteins. These condition number can be conveniently computed owing to the block diagonal structure of M as explained previously. In contrast, computing the condition number, eigenvalues, and singular values of the full matrix A is computationally prohibitive because of the large numbers of elements of these tested proteins. Due to the similarity between A and M, we can use information from M to explain why solving $Ax = b$ on different proteins encounters various convergence speed since we observe that the slow convergence of solving $Ax = b$ implies the slow convergence of solving $My = z$. For example, in the last column of Table 3 large condition numbers (3760.7, 214288.8, 89069.7, 3850.6, 1671.7, 4823.5) for proteins (2pde, 1sh1, 1bpi, 1a7m, 1uv0, 4mth) show large numbers of iterations (100, 110, 110, 67, 56, 69). We highlighted these data in the table.

The diagonal block structure of M make it possible for us to find the singular values conveniently and the computational cost is $O(N_0^3 N/N_0) = O(NN_0^2)$. We choose two proteins (1ajj, 1a63) with good convergence speed (about 10 iterations) and two proteins (2sh1, 2pde) with poor convergence speed (100+ iterations) and plot all singular values of matrix M in Fig. 5. The plot can be used to explain the convergence speed. On one hand, the ratios between the largest and smallest singular values, the 2-norm condition numbers, for proteins 1ajj and 1a63 are much smaller than those from proteins 1sh1 and 1a63. On the other hand, the singular values of proteins 1ajj and 1a63 cluster away from the origin while the singular values of proteins 1ajj and 1a63 cluster near the origin. The computed singular values from blocks also enable us to identify at which leaf the interactions affect the matrix condition the most. Our finding is the largest and smallest singular values mostly appear in the same block/leaf, in which some elements are unusually close. Note matrix A, so is M, is designed to be diagonally dominant using the well-posed integral form [29], but these close elements (small $|\mathbf{x} - \mathbf{y}|$) make large kernels K_{1-4}, particularly the K_2 and K_3 kernels, which are in the blue and black closer-to-diagonal regions as shown in Fig. 3, resulting in deteriorated condition number. This finding again confirms our conjecture that it is the triangulation quality that slows down the GMRES convergence.

We also reported the condition numbers of A and $M^{-1}A$ for some selected cases in Table 4. In order to do that, we explicitly output the matrices and use Matlab to calculate the condition numbers. Note the problem size is too big for calculating condition numbers for large proteins thus we only provide results for some small proteins at small densities. From the table, we can see the block diagonal preconditioning scheme significantly reduces the condition number thus requires much less number of iterations for convergence.

3.4. Preconditioning performance on molecular surfaces generated by other mesh generators

In addition to MSMS, there are many other triangular mesh generators such as NanoShaper [69], TMSmesh [70], ESES [71], etc. Theoretically, the Fredholm second kind integral form is well-conditioned as ϕ_1 and $\frac{\partial \phi_1}{\partial \nu_{\mathbf{y}}}$ term at the LHS of Eqs. (12a)–(12b) enforce the "diagonally dominant" after the discretization. However, numerically poor triangulation is unavoidable when molecular surfaces of some complex proteins are triangulated. To this end, our preconditioning scheme

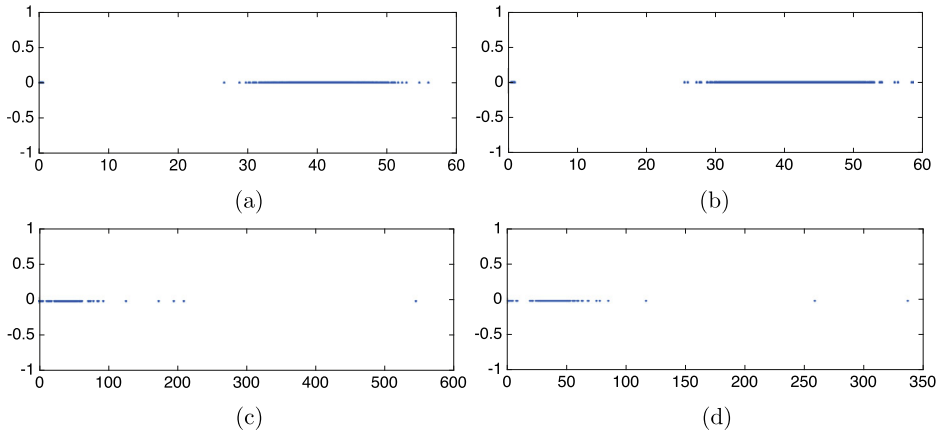

Fig. 5. Singular values plots of the M matrices: (a) 1ajj, (b) 1a63, (c) 1sh1, (d) 2pde.

Table 4
Condition numbers of A and $M^{-1}A$ and iteration numbers when solving PB equation using TABI with diagonal precondition (d) and block diagonal precondition (bd) for a few small proteins at MSMS density $d = 1$.

PDB	# of ele.	# of it. (d)	# of it. (bd)	cond. (A)	cond. ($M^{-1}A$)
1ajj	6027	12	8	411.2	335.8
2pde	6582	42	12	3950.0	651.1
1sh1	7657	29	10	2376.7	938.5

Table 5
Convergence performance for selected proteins using NanoShaper for triangulation; "d" stands for diagonal preconditioning and "bd" stands for block diagonal preconditioning.

PDB	# of ele.	E_{sol} (kcal/mol)		# of its.		CPU times		
		d	bd	d	bd	d	bd	ratio
4dpf	43100	−21180.92	−21181.75	77	12	1069.2	197.2	5.42
1hg8	306852	−13175.01	−13174.30	27	11	3880.2	1780.8	2.18
3sqe	572352	−17080.73	−17079.44	49	16	15707.1	5511.1	2.85
2cek	875828	−26957.91	−26957.84	26	16	13421.3	9061.8	1.48
3lod	335064	−8796.08	−8795.40	73	14	12480.8	2674.9	4.67

should benefit TABI solvers using various mesh generators. To see this, we tested some cases using NanoShaper as the molecular surface generator, which requires large number of iterations [17]. The results as included in Table 5 shows significant improvements when the block diagonal preconditioning scheme is used as opposed the diagonal preconditioning, which demonstrates the broader application of our preconditioning schemes.

4. Conclusion

In this paper, we present a newly discovered block diagonal preconditioning scheme to improve the TABI Poisson–Boltzmann solver, which previously use diagonal preconditioning to accelerate GMRES solver. The block diagonal preconditioning algorithm is simple, efficient, robust, and accurate, which cancels the slow-down effects caused by the mesh quality and the added computational cost due to preconditioning is insignificant. Our simulation on various proteins shows that for many problems with previously slow convergence now converges as quick as cases without GMRES convergence issues. This improvement is consistent with the refinement of the triangulation mesh. Through numerical simulations, we also suggest the optimal treecode parameters to use TABI solver updated with this new preconditioning scheme. In addition, with this preconditioning scheme, the cost of preprocessing mesh in deleting poor quality triangles can be greatly reduced. We believe this simple preconditioning idea and scheme can benefit many multipole methods accelerated boundary integral Poisson–Boltzmann solvers such as [34–44]. This idea and its variation can also be used to accelerate solving boundary integral equations originated from broader areas such as scattering, fluids, elasticity, etc.

Acknowledgements

The work was supported by NSF grant DMS-1418957, SMU new faculty startup fund and SMU Center for Scientific Computing. The authors thank Professor Robert Krasny for useful suggestions.

References

[1] W. Geng, R. Krasny, A treecode-accelerated boundary integral Poisson–Boltzmann solver for electrostatics of solvated biomolecules, J. Comput. Phys. 247 (2013) 62–78.
[2] W. Geng, F. Jacob, A GPU-accelerated direct-sum boundary integral Poisson–Boltzmann solver, Comput. Phys. Commun. 184 (6) (2013) 1490–1496.
[3] T. Schlick, Molecular Modeling and Simulation: An Interdisciplinary Guide, Springer, 2010.
[4] N.A. Baker, Improving implicit solvent simulations: a Poisson-centric view, Curr. Opin. Struct. Biol. 15 (2) (2005) 137–143.
[5] V. Cherezov, D.M. Rosenbaum, M.A. Hanson, S.G.F. Rasmussen, F.S. Thian, T.S. Kobilka, H.-J. Choi, P. Kuhn, W.I. Weis, B.K. Kobilka, R.C. Stevens, High-resolution crystal structure of an engineered human 2-adrenergic g protein-coupled receptor, Science 318 (5854) (2007) 1258–1265.
[6] F. Dong, M. Vijaykumar, H.X. Zhou, Comparison of calculation and experiment implicates significant electrostatic contributions to the binding stability of barnase and barstar, Biophys. J. 85 (1) (2003) 49–60.
[7] N. Huang, Y. Chelliah, Y. Shan, C.A. Taylor, S.-H. Yoo, C. Partch, C.B. Green, H. Zhang, J.S. Takahashi, Crystal structure of the heterodimeric CLOCK:BMAL1 transcriptional activator complex, Science 337 (6091) (2012) 189–194.
[8] 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.
[9] E. Alexov, E.L. Mehler, N. Baker, A.M. Baptista, Y. Huang, F. Milletti, J.E. Nielsen, D. Farrell, T. Carstensen, M.H.M. Olsson, J.K. Shen, J. Warwicker, S. Williams, J.M. Word, Progress in the prediction of pka values in proteins, Proteins 79 (Dec. 2011) 3260–3275.
[10] J. Antosiewicz, J.A. McCammon, M.K. Gilson, The determinants of pK_as in proteins, Biochemistry 35 (24) (1996) 7819–7833.
[11] J. Hu, S. Zhao, W. Geng, Accurate pka computation using matched interface and boundary (MIB) method based Poisson–Boltzmann solver, Commun. Comput. Phys. 2 (2018) 520–539.
[12] J.E. Nielsen, J.A. McCammon, Calculating pka values in enzyme active sites, Protein Sci. 12 (9) (2003) 1894–1901.
[13] Y.C. Zhou, B. Lu, A.A. Gorfe, Continuum electromechanical modeling of protein-membrane interactions, Phys. Rev. E 82 (Oct. 2010) 041923.
[14] 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 (09 2010) 1–12.
[15] C. García-García, D.E. Draper, Electrostatic interactions in a peptide–RNA complex, J. Mol. Biol. 331 (1) (2003) 75–88.
[16] D.D. Nguyen, B. Wang, G.-W. Wei, Accurate, robust, and reliable calculations of Poisson–Boltzmann binding energies, J. Comput. Chem. 38 (13) (2017) 941–948.
[17] L. Wilson, R. Krasny, A comparison of two molecular surface triangulation codes in a boundary integral Poisson–Boltzmann framework, 2018, in preparation.
[18] T. Simonson, G. Archontis, M. Karplus, Free energy simulations come of age: protein–ligand recognition, Acc. Chem. Res. 35 (6) (2002) 430–437, PMID: 12069628.
[19] J.A. Wagoner, N.A. Baker, Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms, Proc. Natl. Acad. Sci. 103 (22) (2006) 8331–8336.
[20] N. Unwin, Refined structure of the nicotinic acetylcholine receptor at 4 Å resolution, J. Mol. Biol. 346 (4) (2005) 967–989.
[21] N.A. Baker, Poisson–Boltzmann methods for biomolecular electrostatics, Methods Enzymol. 383 (2004) 94–118.
[22] Q. Lu, R. Luo, A Poisson–Boltzmann dynamics method with nonperiodic boundary condition, J. Chem. Phys. 119 (21) (2003) 11035–11047.
[23] W. Im, D. Beglov, B. Roux, Continuum solvation model: electrostatic forces from numerical solutions to the Poisson–Boltzmann equation, Comput. Phys. Commun. 111 (1–3) (1998) 59–75.
[24] B. Honig, A. Nicholls, Classical electrostatics in biology and chemistry, Science 268 (5214) (1995) 1144–1149.
[25] W. Geng, S. Yu, G.W. Wei, Treatment of charge singularities in implicit solvent models, J. Chem. Phys. 127 (2007) 114106.
[26] S. Yu, W. Geng, G.W. Wei, Treatment of geometric singularities in implicit solvent models, J. Chem. Phys. 126 (2007) 244108.
[27] L.F. Greengard, J. Huang, A new version of the fast multipole method for screened Coulomb interactions in three dimensions, J. Comput. Phys. 180 (2) (2002) 642–658.
[28] P. Li, H. Johnston, R. Krasny, A Cartesian treecode for screened Coulomb interactions, J. Comput. Phys. 228 (10) (2009) 3858–3868.
[29] 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.
[30] J. Chen, W. Geng, Parallel computing of the adaptive n-body treecode algorithm for solving boundary integral Poisson–Boltzmann equation, Lect. Notes Comput. Sci. 9576 (2015) 1–9.
[31] N.A. Baker, D. Sept, M.J. Holst, J.A. Mccammon, The adaptive multilevel finite element solution of the Poisson–Boltzmann equation on massively parallel computers, IBM J. Res. Dev. 45 (3–4) (2001) 427–438.
[32] E. Jurrus, D. Engel, K. Star, K. Monson, J. Brandi, L.E. Felberg, D.H. Brookes, L. Wilson, J. Chen, K. Liles, M. Chun, P. Li, D.W. Gohara, T. Dolinsky, R. Konecny, D.R. Koes, J.E. Nielsen, T. Head-Gordon, W. Geng, R. Krasny, G.-W. Wei, M.J. Holst, J.A. McCammon, N.A. Baker, Improvements to the APBS biomolecular solvation software suite, Protein Sci. 27 (2018) 112–128.
[33] M.F. Sanner, A.J. Olson, J.C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.
[34] B.J. Yoon, A.M. Lenhoff, A boundary element method for molecular electrostatics with electrolyte effects, J. Comput. Chem. 11 (9) (1990) 1080–1086.
[35] J. Liang, S. Subranmaniam, Computation of molecular electrostatics with boundary element methods, Biophys. J. 73 (1997) 1830–1841.
[36] A.H. Boschitsch, M.O. Fenley, H.-X. Zhou, Fast boundary element method for the linear Poisson–Boltzmann equation, J. Phys. Chem. B 106 (10) (2002) 2741–2754.
[37] B. Lu, X. Cheng, J.A. McCammon, A new-version-fast-multipole-method accelerated electrostatic calculations in biomolecular systems, J. Comput. Phys. 226 (2) (2007) 1348–1366.
[38] M.D. Altman, J.P. Bardhan, J.K. White, B. Tidor, Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized Poisson–Boltzmann equation with curved boundary elements, J. Comput. Chem. 30 (1) (2009) 132–153.
[39] L. Greengard, D. Gueyffier, P.-G. Martinsson, V. Rokhlin, Fast direct solvers for integral equations in complex three-dimensional domains, Acta Numer. 18 (2009) 005.
[40] 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 (2) (2011) 826–848.
[41] W. Geng, Parallel higher-order boundary integral electrostatics computation on molecular surfaces with curved triangulation, J. Comput. Phys. 241 (2013) 253–265.
[42] B. Zhang, B. Lu, X. Cheng, J. 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 (1) (2013) 107–128.
[43] Q. Sun, E. Klaseboer, D.Y.C. Chan, A robust and accurate formulation of molecular and colloidal electrostatics, J. Chem. Phys. 145 (5) (2016) 054106.
[44] Y. Zhong, K. Ren, R. Tsai, An implicit boundary integral method for computing electric potential of macromolecules in solvent, J. Comput. Phys. 359 (2018) 199–215.
[45] M.E. Davis, J.D. Madura, J. Sines, B.A. Luty, S.A. Allison, J.A. McCammon, Diffusion-controlled enzymatic reactions, Methods Enzymol. 202 (1991) 473–497.

[46] 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 (18) (2001) 10037–10041.
[47] R. Luo, L. David, M.K. Gilson, Accelerated Poisson–Boltzmann calculations for static and dynamic systems, J. Comput. Chem. 23 (13) (2002) 1244–1253.
[48] Q. Cai, J. Wang, H.-K. Zhao, R. Luo, On removal of charge singularity in Poisson–Boltzmann equation, J. Chem. Phys. 130 (14) (2009).
[49] D. Chen, Z. Chen, C. Chen, W. Geng, G.W. Wei, MIBPB: a software package for electrostatic analysis, J. Comput. Chem. 32 (2011) 657–670.
[50] W. Deng, X. Zhufu, J. Xu, S. Zhao, A new discontinuous Galerkin method for the nonlinear Poisson–Boltzmann equation, Appl. Math. Lett. 257 (2015) 1000–1021.
[51] J. Ying, D. Xie, A new finite element and finite difference hybrid method for computing electrostatics of ionic solvated biomolecule, J. Comput. Phys. 298 (2015) 636–651.
[52] R.J. Zauhar, R.S. Morgan, A new method for computing the macromolecular electric potential, J. Mol. Biol. 186 (4) (1985) 815–820.
[53] B.Z. Lu, Y.C. Zhou, M.J. Holst, J.A. McCammon, Recent progress in numerical methods for the Poisson–Boltzmann equation in biophysical applications, Commun. Comput. Phys. 3 (5) (2008) 973–1009.
[54] W. Rocchia, S. Sridharan, A. Nicholls, E. Alexov, A. Chiabrera, B. Honig, Rapid grid-based construction of the molecular surface and the use of induced surface charge to calculate reaction field energies: applications to the molecular systems and geometric objects, J. Comput. Chem. 23 (1) (2002) 128–137.
[55] S. Hou, W. Wang, L. Wang, Numerical method for solving matrix coefficient elliptic equation with sharp-edged interfaces, J. Comput. Phys. 229 (19) (2010) 7162–7179.
[56] Z. Li, H. Ji, X. Chen, Accurate solution and gradient computation for elliptic interface problems with variable coefficients, SIAM J. Numer. Anal. 55 (2) (2017) 570–597.
[57] D. Xie, New solution decomposition and minimization schemes for Poisson–Boltzmann equation in calculation of biomolecular electrostatics, J. Comput. Phys. 275 (2014) 294–309.
[58] L. Wang, S. Hou, L. Shi, A numerical method for solving three-dimensional elliptic interface problems with triple junction points, Adv. Comput. Math. 44 (1) (2018) 175–193.
[59] B. Lu, X. Cheng, J. Huang, J.A. McCammon, Order N algorithm for computation of electrostatic interactions in biomolecular systems, Proc. Natl. Acad. Sci. 103 (51) (2006) 19314–19319.
[60] J. Barnes, P. Hut, A hierarchical O(N log N) force-calculation algorithm, Nature 324 (12) (1986) 446–449.
[61] Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.
[62] Z.-H. Duan, R. Krasny, An adaptive treecode for computing nonbonded potential energy in classical molecular systems, J. Comput. Chem. 22 (2) (2001) 184–195.
[63] K. Lindsay, R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, J. Comput. Phys. 172 (2) (2001) 879–907.
[64] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D. States, S. Swaminathan, M. Karplus, CHARMM: a program for macromolecular energy, minimization, and dynamics calculations, J. Comput. Chem. 4 (1983) 187–217.
[65] T.J. Dolinsky, P. Czodrowski, H. Li, J.E. Nielsen, J.H. Jensen, G. Klebe, N.A. Baker, PDB2PQR: expanding and upgrading automated preparation of biomolecular structures for molecular simulations, Nucleic Acids Res. 35 (2007).
[66] M.J. Holst, The Poisson–Boltzmann Equation: Analysis and Multilevel Numerical Solution, PhD thesis, UIUC, 1994.
[67] R.E. Lehotzky, C.L. Partch, S. Mukherjee, H.L. Cash, W.E. Goldman, K.H. Gardner, L.V. Hooper, Molecular basis for peptidoglycan recognition by a bactericidal lectin, Proc. Natl. Acad. Sci. 107 (17) (2010) 7722–7727.
[68] S. Mukherjee, H. Zheng, M.G. Derebe, K.M. Callenberg, C.L. Partch, D. Rollins, D.C. Propheter, J. Rizo, M. Grabe, Q.-X. Jiang, L.V. Hooper, Antibacterial membrane attack by a pore-forming intestinal C-type lectin, Nature 505 (01 2014) 103–107.
[69] S. Decherchi, W. Rocchia, A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale, PLoS ONE 8 (04 2013) 1–15.
[70] M. Chen, B. Lu, Tmsmesh: a robust method for molecular surface mesh generation using a trace technique, J. Chem. Theory Comput. 7 (1) (2011) 203–212, PMID: 26606233.
[71] B. Liu, B. Wang, R. Zhao, Y. Tong, G.-W. Wei, ESES: software for Eulerian solvent excluded surface, J. Comput. Chem. 38 (7) (2017) 446–466.