Journal of Computational Physics 478 (2023) 111981

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A Cartesian FMM-accelerated Galerkin boundary integral Poisson-Boltzmann solver

Jiahui Chen [a], Johannes Tausch [b], Weihua Geng [b,*]

[a] Department of Mathematics, Michigan State University, East Lansing, MI 48824, USA
[b] Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

ARTICLE INFO

Article history:
Received 4 August 2022
Received in revised form 20 January 2023
Accepted 27 January 2023
Available online 2 February 2023

Keywords:
Fast multipole method
Electrostatic
Boundary integral
Poisson-Boltzmann
Preconditioning
GMRES

ABSTRACT

The Poisson-Boltzmann model is an effective and popular approach for modeling solvated biomolecules in continuum solvent with dissolved electrolytes. In this paper, we report our recent work in developing a Galerkin boundary integral method for solving the linear Poisson-Boltzmann (PB) equation. The solver has combined advantages in accuracy, efficiency, and memory usage as it applies a well-posed boundary integral formulation to circumvent many numerical difficulties associated with the PB equation and uses an $O(N)$ Cartesian Fast Multipole Method (FMM) to accelerate the GMRES iteration. In addition, special numerical treatments such as adaptive FMM order, block diagonal preconditioners, Galerkin discretization, and Duffy's transformation are combined to improve the performance of the solver, which is validated on benchmark Kirkwood's sphere and a series of testing proteins.

1. Introduction

In biomolecular simulations, electrostatic interactions are of paramount importance due to their ubiquitous existence and significant contribution in the force fields, which governs the dynamics of molecular simulation. However, computing non-bonded interactions is challenging since these pairwise interactions are long-range with $O(N^2)$ computational cost, which could be prohibitively expensive for large systems. To reduce the degree of freedom of the system in terms of electrostatic interactions, an implicit solvent Poisson-Boltzmann (PB) model is used. In this model, the explicit water molecules are treated as continuum and the dissolved electrolytes are approximated using the statistical Boltzmann distribution. The PB model has broad applications in biomolecular simulations such as protein structure [1], chromatin packing [2], pKa [3–5], membrane [6], binding energy [7], solvation free energy [8], ion channel profiling [9], etc.

The PB model is an elliptic interface problem with several numerical difficulties such as discontinuous dielectric coefficients, singular sources, a complex interface, and unbounded domains. Grid-based finite difference or finite volume discretization that discretize the entire volumetric domain have been developed in, e.g., [10–15]. The grid-based discretization is efficient and robust and is therefore popular. However, solvers that are based on discretizing the partial differential equation 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 [16,17] and singularity [18–22] treatments are

* Corresponding author at: Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA.
E-mail address: wgeng@smu.edu (W. Geng).

https://doi.org/10.1016/j.jcp.2023.111981
0021-9991/

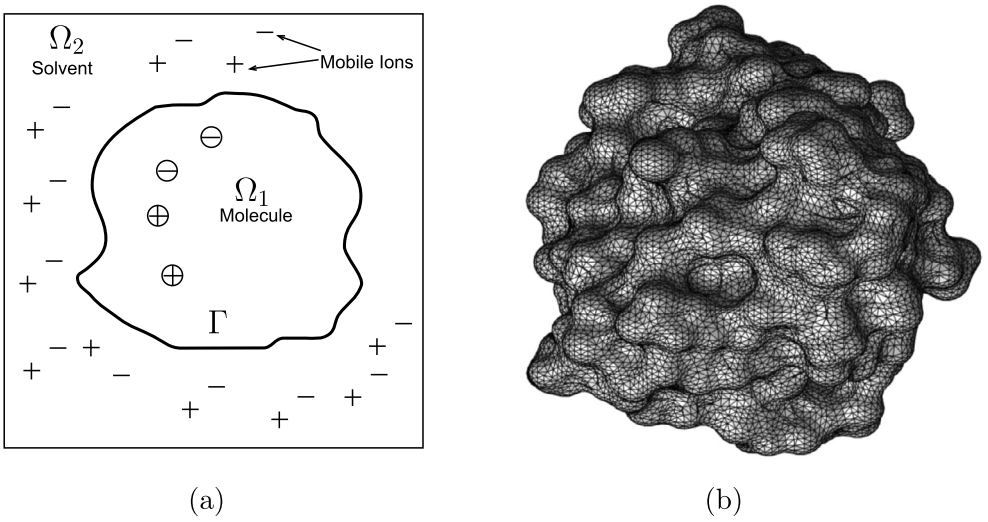

Fig. 1. Schematic models; (a) the PB implicit solvent model, in which the molecular surface Γ separates space into the solute region Ω_1 and solvent region Ω_2; (b) the triangulation of molecular surface of protein Barstar at MSMS [42] density $d = 5$ (# of vertices per Å^2).

applied. These treatments come at the price of more complicated discretization scheme and possibly reduced convergence speed of the iterative solver for the linear system.

An alternative approach is to reformulate the linear PB equation as a boundary integral equation and use the boundary elements to discretize the molecular surface, e.g. [23–31]. Besides the reduction from three dimensional space to the two dimensional molecular boundary, this approach has the advantage that singular charges, interface conditions, and far-field condition are incorporated analytically in the formulation, and hence do not impose additional approximation errors.

One drawback of the boundary integral method is that the discretization leads to a dense matrix with $O(N^2)$ complexity for storage and matrix-vector product, which is prohibitively expensive when N is large. Fortunately, iterative solvers can be accelerated by fast methods such as fast multipole methods (FMM) [24–26,32,33] and treecodes [34–36]. Our recently developed treecode-accelerated boundary integral (TABI) Poisson-Boltzmann solver [28] is an example of a code that combines 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 [23] and the $O(N \log N)$ treecode [35]. It also has advantages in memory use and parallelization [28,37]. The TABI solver has been used by many computational biophysics/biochemistry groups and it has been disseminated as a standalone code [28] or as a contributing module of the popular APBS software package [38,39].

Recently, based on feedback from TABI solver users and our gained experience in theoretical development and practical applications, we realized that we could still improve the TABI solver in the following aspects. First, the $O(N \log N)$ treecode can be replaced by the $O(N)$ FMM method with manageable extra costs in memory usage and complexity of the algorithms. Second, the singularity that occurs when the Poisson's or Yukawa's kernel is evaluated was previously handled by simply removing the singular triangle [25,28] in fact can be treated by using the Duffy transformation [40] analytically, achieving improved accuracy. Third, the collocation scheme used in TABI solver can be updated by using Galerkin discretization with further advantage in maintaining desired accuracy. Fourth, the treecode-based preconditioning scheme that was used in TABI solver [41] can be similarly developed and used under the FMM frame, receiving significant improvement in convergence and robustness. By combining all these new features, we developed a Cartesian fast multipole method (FMM) accelerated Galerkin boundary integral (FAGBI) Poisson–Boltzmann solver. In the remainder of this article, we provide more detail about the theoretical background of the numerical algorithms related to the FAGBI solver. We conclude with a discussion of the numerical results obtained with our implementation.

2. Theory and algorithms

In this section we briefly describe the Poisson-Boltzmann (PB) implicit solvent model and review the boundary integral form of the PB equation and its Galerkin discretization. Based on this background information, we then provide details of our recently developed FMM-accelerated Galerkin boundary integral (FAGBI) Poisson-Boltzmann solver, which involves the boundary integral form, multipole expansion scheme, and a block diagonal 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. Note our choice of the molecular surface is the Solvent Excluded Surface (SES) as

the trace of the solvent probe (e.g. water molecule as a sphere with radius 1.4 Å) when it rolls while contacting the solute atoms. This surface is in general C^1 continuous but singularities and cusps can present under some extreme configurations. Fig. 1(b) is an example of the molecular surface Γ as the triangulated surface of protein barstar [43]. In domain Ω_1, the solute is represented by N_c partial charges q_k located at atomic centers $\mathbf{x}_k$ 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 for the boundary integral approach. 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 I_s; $\bar{\kappa} = 0$ 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 condition,

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

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. Boundary integral form of PB model

This section summarizes the well-conditioned boundary integral form of the PB implicit solvent model we employ [23, 28]. Applying Green's second identity and properties of fundamental solutions to Eq. (1) yields the electrostatic potential in each domain [23],

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

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

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

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

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 (the subscript 1 denotes the inside domain) and its normal derivative $\partial\phi_1/\partial\nu$ on Γ, [23,28],

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

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

where $\varepsilon = \varepsilon_2/\varepsilon_1$, and the kernels $K_{1,2,3,4}$ and source terms $S_{1,2}$ are

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

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

and

$$S_1(\mathbf{x}) = \frac{1}{\varepsilon_1} \sum_{k=1}^{N_c} q_k G_0(\mathbf{x}, \mathbf{x}_k), \quad S_2(\mathbf{x}) = \frac{1}{\varepsilon_1} \sum_{k=1}^{N_c} q_k \frac{\partial G_0(\mathbf{x}, \mathbf{x}_k)}{\partial \nu_\mathbf{x}}. \tag{8}$$

As given in Eqs. (7a)-(7b) and (8), 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 [23,28]. Since the Coulomb potential is singular, the kernels have the following behavior

$$K_1(\mathbf{x},\mathbf{y}) = O(1),\ K_{2,3,4}(\mathbf{x},\mathbf{y}) = O\left(\frac{1}{|\mathbf{x}-\mathbf{y}|}\right),$$

as $\mathbf{y}\to\mathbf{x}$.

After the potentials ϕ_1 and $\partial\phi_1/\partial\nu$ have been found by solving the boundary integral equation, the electrostatic solvation free energy can be obtained by

$$E_{\text{sol}} = \frac{1}{2}\sum_{k=1}^{N_c} q_k\phi_{\text{reac}}(\mathbf{x}_k) = \frac{1}{2}\sum_{k=1}^{N_c} q_k \int_\Gamma \left[K_1(\mathbf{x}_k,\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu} + K_2(\mathbf{x}_k,\mathbf{y})\phi_1(\mathbf{y})\right] dS_{\mathbf{y}}, \tag{9}$$

where $\phi_{\text{reac}}(\mathbf{x}) = \phi_1(\mathbf{x}) - S_1(\mathbf{x})$ is the reaction potential [23,28].

2.3. Galerkin discretization

In solving the boundary integral PB equation, both the molecular surface and the solution function need to be discretized. The molecular surface Γ (e.g. SES) is usually approximated by a collection of triangles

$$\Gamma_N = \bigcup_{i=1}^{N} \tau_i, \tag{10}$$

where N is number of elements and τ_i for $i = 1,\dots,N$ is a planar triangular boundary element with mid-point $\mathbf{x}_i^c$. This triangulation must be conforming, i.e., the intersection of two different triangles is either empty, or a common vertex or edge. Fortunately, algorithm and software for generating are available [42,44,45], and our choice for our computations is MSMS [42]. Here, the resolution of the surface can be controlled by the parameter d that controls the number of vertices per Å^2. For example, Fig. 1 (b) shows the triangulated molecule surface of the protein barstar, which will bind another protein barnase to form a biomolecular complex (PDB: 1b2s) [43].

Each triangle τ_i of Γ_N is the parametric image of the reference triangle τ

$$\tau = \left\{\xi = (\xi_1,\xi_2)\in\mathbb{R}^2 : 0\le\xi_1\le 1, 0\le\xi_2\le\xi_1\right\}. \tag{11}$$

If $\mathbf{u}_i$, $\mathbf{v}_i$ and $\mathbf{w}_i$ are the vertices then the parameterization is given by

$$\mathbf{x}(\xi) = \mathbf{u}_i + \xi_1(\mathbf{v}_i-\mathbf{u}_i) + \xi_2(\mathbf{w}_i-\mathbf{u}_i)\in\tau_i \quad \text{for } \xi=(\xi_1,\xi_2)\in\tau. \tag{12}$$

The area of the element, the local mesh size, and the global mesh size of the boundary elements τ_i are given as $A_i = \frac{1}{2}|(\mathbf{v}_i-\mathbf{u}_i)\times(\mathbf{w}_i-\mathbf{u}_i)|$, $h_i = \sqrt{A_i}$, and $h = \max_{1\le i\le N} h_i$.

Since a function f defined on τ_i can be interpreted as a function $g(\xi)$ with respect to the reference element τ,

$$f(\mathbf{x}) = f(\mathbf{x}(\xi)) = g(\xi) \quad \text{for } \xi\in\tau, \quad \mathbf{x}\in\tau_i, \tag{13}$$

we can define a finite element space by functions on Γ_N whose pullbacks to the reference triangle are polynomials in ξ. The simplest example is the space of piecewise constant functions, which are polynomials of order zero on each triangle, which will be denoted by $S_h^0(\Gamma_N)$. Obviously, the dimension of this space is N and the basis is given by the box functions

$$\psi_i^0(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}\in\tau_{\mathbf{i}}, \\ 0 & \text{otherwise}, \end{cases} \tag{14}$$

where i is an index of a triangle.

The next step up are piecewise linear functions. Since there are three independent linear functions on τ, namely,

$$\psi_1^1(\xi) = 1-\xi_1, \quad \psi_2^1(\xi) = \xi_1-\xi_2, \quad \psi_3^1(\xi) = \xi_2 \quad \text{for } \xi=(\xi_1,\xi_2)\in\tau, \tag{15}$$

the dimension is $3N$. Usually, one works with the space of continuous linear functions, denoted by $S_h^1(\Gamma_N)$. It is not hard to see that the dimension of this space is the number of vertices and that the basis is given by

$$\psi_i^c(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{x} = \mathbf{v}_i, \\ 0 & \text{for } \mathbf{x} = \mathbf{v}_j \ne \mathbf{v}_i, \\ \text{piecewise linear} & \text{elsewhere}, \end{cases} \tag{16}$$

where v_i is the i-th vertex.

The approximation powers of piecewise polynomial spaces are well known, see, e.g., [46]. For a function $w \in H^1_{pw}(\Gamma_N)$ we denote by w^0_h the L_2-orthogonal projection of w into the space of piecewise constant functions, then

$$\|w - w^0_h\|_{L_2(\Gamma_N)} \le c \left(\sum_{i=1}^N h_i^2 |w|^2_{H^1(\tau_i)} \right)^{\frac{1}{2}} \le ch |w|_{H^1_{pw}(\Gamma_N)}, \tag{17}$$

where c is the upper bound of the mesh ratio $h_{\max}/h_{\min}$, h is the maximal diameter of a triangle and

$$|w|_{H^1_{pw}(\Gamma_N)} = \left(\sum_{i=1}^N |w|^2_{H^1(\tau_i)} \right)^{1/2}. \tag{18}$$

Thus, the constant piecewise basis function can give a convergence rate of maximum $O(h)$.

Likewise, the error for the L_2-orthogonal projection w^1_h of $w \in H^2_{pw}(\Gamma_N)$ into $S^1_h(\Gamma_N)$ is

$$\|w - w^1_h\|_{L_2(\Gamma_N)} \le c \left(\sum_{i=1}^N h_i^2 |w|^2_{H^2(\tau_i)} \right)^{\frac{1}{2}} \le ch^2 |w|_{H^2_{pw}(\Gamma_N)}. \tag{19}$$

Finite element spaces with higher order polynomials could also be considered, however their practical value for surfaces with low regularity and complicated geometries is limited.

The Galerkin discretization is based on a variational formulation of integral equations (6a) and (6b). That is, instead of understanding the equations pointwise for $\mathbf{x} \in \Gamma_N$ the equations are multiplied by test functions ψ and ψ^ν and integrated again over Γ_N. Solving the variational form amounts to finding ϕ_1 and $\partial\phi_1/\partial\nu$ such that

$$\int_{\Gamma_N} \left\{ \frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}) - \int_{\Gamma_N} \left[K_1(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{x},\mathbf{y})\phi_1(\mathbf{y}) \right] dS_{\mathbf{y}} \right\} \psi(\mathbf{x}) dS_{\mathbf{x}} = \int_{\Gamma_N} S_1(\mathbf{x})\psi(\mathbf{x}) dS_{\mathbf{x}}, \tag{20a}$$

$$\int_{\Gamma_N} \left\{ \frac{1}{2}\left(1+\frac{1}{\varepsilon}\right) \frac{\partial \phi_1(\mathbf{x})}{\partial \nu} - \int_{\Gamma_N} \left[K_3(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_4(\mathbf{x},\mathbf{y})\phi_1(\mathbf{y}) \right] dS_{\mathbf{y}} \right\} \psi^\nu(\mathbf{x}) dS_{\mathbf{x}} = \int_{\Gamma_N} S_2(\mathbf{x})\psi^\nu(\mathbf{x}) dS_{\mathbf{x}}, \tag{20b}$$

holds for all test functions ψ, ψ^ν. In the Galerkin method the solution and the test functions are formally replaced by functions in the finite element space. To that end, the unknowns are expanded by basis functions ψ_i (which could be either box or hat functions)

$$\phi \approx \sum_{i=1}^N \phi_i \psi_i, \quad \text{and} \quad \frac{\partial \phi}{\partial \nu} \approx \sum_{i=1}^N \phi_i^\nu \psi_i \tag{21}$$

and integral equations are tested against the basis functions. This leads to the linear system $Ax = b$ where x contains the coefficients in (21) and

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \tag{22}$$

The entries of these block matrices are given as

$$\begin{aligned}
A_{11}(i,j) &= \int_{\Gamma_N} \frac{1}{2}(1+\varepsilon)\psi_i(\mathbf{x})\psi_j(\mathbf{x}) \mathrm{d}S_{\mathbf{x}} + \int_{\Gamma_N}\int_{\Gamma_N} K_2(\mathbf{x},\mathbf{y})\psi_i(\mathbf{x})\psi_j(\mathbf{y}) \mathrm{d}S_{\mathbf{y}} \mathrm{d}S_{\mathbf{x}} \\
A_{12}(i,j) &= \int_{\Gamma_N}\int_{\Gamma_N} K_1(\mathbf{x},\mathbf{y})\psi_i(\mathbf{x})\psi_j(\mathbf{y}) \mathrm{d}S_{\mathbf{y}} \mathrm{d}S_{\mathbf{x}} \\
A_{21}(i,j) &= \int_{\Gamma_N}\int_{\Gamma_N} K_4(\mathbf{x},\mathbf{y})\psi_i(\mathbf{x})\psi_j(\mathbf{y}) \mathrm{d}S_{\mathbf{y}} \mathrm{d}S_{\mathbf{x}} \\
A_{22}(i,j) &= \int_{\Gamma_N} \frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\psi_i(\mathbf{x})\psi_j(\mathbf{x}) \mathrm{d}S_{\mathbf{x}} + \int_{\Gamma_N}\int_{\Gamma_N} K_3(\mathbf{x},\mathbf{y})\psi_i(\mathbf{x})\psi_j(\mathbf{y}) \mathrm{d}S_{\mathbf{y}} \mathrm{d}S_{\mathbf{x}}
\end{aligned} \tag{23}$$

and the right hand side is

$$b_1(i) = \int_{\Gamma_N} S_1(\mathbf{x})\psi_i(\mathbf{x})dS_{\mathbf{x}} \quad \text{and} \quad b_2(i) = \int_{\Gamma_N} S_2(\mathbf{x})\psi_i(\mathbf{x})dS_{\mathbf{x}} \tag{24}$$

Since the basis functions vanish on most triangles, the integrations for the coefficients are only local and computed on the fly. For instance, for piecewise constant elements, the integral $\int_\Gamma \dots \psi_i dS_{\mathbf{x}}$ reduces to $\int_{\tau_i} \dots dS_{\mathbf{x}}$. Since the coefficients cannot be expressed in analytical form they have to be calculated by a suitable choice of quadrature rule. However singularities will appear if triangles τ_i and τ_j are identical or sharing common edges and vertices. To overcome this issue, we apply the singularity removing transformation of [47]. This results smooth integrals over a four dimensional cube. The latter integrals are then approximated by tensor product Gauss-Legendre quadrature.

After the solution of the linear system has been obtained, the electrostatic free solvation energy can be calculated using the approximations for the surface potentials and its normal derivative

$$E_{\text{sol}} = \frac{1}{2}\sum_{n=1}^{N_c} q_n \sum_{i=1}^{N} \int_{\tau_i} \left[K_1(\mathbf{x}_n, \mathbf{x})\phi_{1i}^{\nu} + K_2(\mathbf{x}_n, \mathbf{x})\phi_{1i} \right] dS_{\mathbf{x}}. \tag{25}$$

Since the matrix A is a dense and non-symmetric our choice of solver is the GMRES method. In each step of GMRES iteration, a matrix-vector product is calculated and a direct summation for this requires $O(N^2)$ complexity. Below we will introduce the $O(N)$ Cartesian Fast Multipole Method (FMM) to accelerate the matrix-vector product. Calculating the electrostatic solvation free energy E_{sol} in Eq. (25) is $O(N_c N)$ and we use a Cartesian treecode to reduce the cost to $O(N_c \log(N))$. Both FMM and treecode algorithm are described the next for comparison and for the reason that both are used to accelerate the N-body particle-particle interactions.

2.4. Cartesian fast multipole method (FMM)

In this section, we introduce the Cartesian FMM to evaluate matrix vector products with the matrix in (22) efficiently. Instead of multipole expansions, our implementation is based on truncated Taylor series to approximate the Coulomb ($\kappa = 0$) and screened Coulomb ($\kappa \neq 0$) potentials. For these potentials there exists a simple recurrence relationship to compute all derivatives up to order p in $O(p^4)$ operations, see [48]. This considerably simplifies the translation operators for the kernels K_{1-4} because they involve different values of κ. Furthermore, the moment-to-moment (MtM) and local-to-local (LtL) translation are easily derived using the binomial formula.

When a refined mesh is required for a larger N, increasing the expansion order is essential to control the accuracy. The FMM error analysis implies that the truncated Taylor expansion error has the same magnitude as the discretization error if the expansion order is adjusted to the level according to the formula

$$p_l = p_L + L - l \tag{26}$$

where $l = 0, 1, \cdots, L$ with $l = 0$ the coarsest level and $l = L$ the finest level. That is, the finest level uses a low-order expansion p_L, and the order is incremented in each coarser level, see [33].

Note that the multipole series is more efficient as it contains $(p+1)^2$ terms, while the Taylor series has $p(p+1)(p+2)/6$ terms. This difference becomes significant with larger values of p. However, with the variable order scheme the advantage of the multipole series becomes negligible because most translation operators are in the fine levels where the number of terms in both series are comparable.

The matrix vector product can be considered as generalized N-body problem of the form

$$V_i = \int_{\Gamma_N}\int_{\Gamma_N} \psi_i(\mathbf{x}) \frac{\partial^l}{\partial \nu_{\mathbf{x}}^l} \frac{\partial^k}{\partial \nu_{\mathbf{y}}^k} G(\mathbf{x}, \mathbf{y}) f_h(\mathbf{y}) dS_{\mathbf{y}} dS_{\mathbf{x}}, \tag{27}$$

where $k, l \in \{0, 1\}$ and f_h is a linear combination of the basis functions ψ_j.

Next we show how the Cartesian FMM is used under the framework of boundary element method.

Fig. 2a is a 2-D illustration of a discretized molecular surface Γ_N embedded in a hierarchy of cubes (squares in the image). Each black solid dot represents a boundary triangle τ_i for $i = 1, \cdots, N$.

A cluster c in any level l is defined as the union of triangles whose centroid are located in a cube of that level. C_l is the set of all clusters in level l.

The level-0 cube is the smallest axiparallel cube that contains Γ_N and thus $C_0 = \Gamma_N$. The refinement of coarser cubes into finer cubes stops when clusters in the finest level contain at most a predetermined (small) number of triangles. For a cluster c we denote by B_c the smallest axiparallel rectangular box that contains c and write $\mathbf{x}_c$ for its center and ρ_c for the half-length of its longest diagonal. Note that B_c can be considerably smaller than its cube, this is why we call this process the shrink scheme. For two clusters c and c' in the same level we denote by

$$\eta(c, c') = \frac{\rho_c + \rho_{c'}}{|\mathbf{x}_c - \mathbf{x}_{c'}|} \tag{28}$$

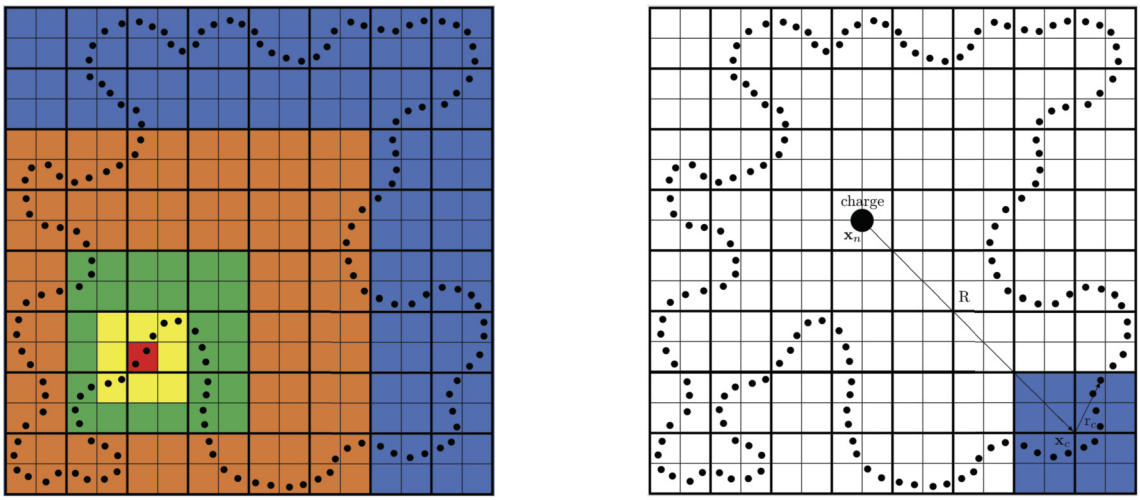

Fig. 2. FMM vs Treecode structure. Left: FMM cluster-cluster interaction list; Right: treecode particle-cluster interaction (R is the distance from the charge to the blue cluster's center; r_c is the radius of the blue cluster c which is the farthest particle inside c to the center of c. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the separation ratio of the two clusters. This number determines the convergence rate of the Taylor series expansion, see [33]. Two clusters in the same level are neighbors if their separation ratio is larger than a predetermined constant. $\mathcal{N}(c)$ denotes the set of its neighbors for a given cluster c. The set of nonempty children of c that are generated in the refinement process is denoted by $\mathcal{K}(c)$. Finally, we use $\mathcal{I}(c)$ to denote the interaction list for a cluster c, which are clusters at the same level such that for any $c' \in \mathcal{I}(c)$, the parent of c' is a neighbor of the parent of c, but c' itself is not a neighbor of c.

Under the FMM framework, the evaluation of Eq. (27) consists of the near field direct summation and the Taylor expansion approximation for well-separated far field. The near field direct summation happens in between neighboring panels in the finest level. The far field summation is done by multipole or Taylor expansions between interaction lists in all levels. This process is described in many papers, so we do not give details about the derivation.

To emphasize the distinguished features of the Cartesian FMM, we consider a cluster-cluster interaction between two clusters c and $c' \in \mathcal{I}(c)$. Let u be the potential due to sources in c' which is evaluated in c, then by Taylor expansion of the kernel with center $\mathbf{x} = \mathbf{x}_c$ and $\mathbf{y} = \mathbf{x}_{c'}$ one finds easily that

$$u_{c,c'}(\mathbf{x}) = \int_{c'} \frac{\partial^l}{\partial \nu_{\mathbf{x}}^l} \frac{\partial^k}{\partial \nu_{\mathbf{y}}^k} G(\mathbf{x},\mathbf{y}) f_h(\mathbf{y}) dS_{\mathbf{y}} \approx \sum_{|\alpha| \le p} \lambda_c^\alpha \frac{\partial^l}{\partial \nu_{\mathbf{x}}^l} (\mathbf{x} - \mathbf{x}_c)^\alpha, \tag{29}$$

where and $\alpha = (\alpha_1, \alpha_2, \alpha_3) \in \mathbb{N}^3$ is a multi-index. The expansion coefficients are given by

$$\lambda_c^\alpha = \sum_{|\beta|=0}^{p-|\alpha|} \frac{D^{\alpha+\beta} G(\mathbf{x}_c, \mathbf{x}_{c'})}{\alpha!\beta!} (-1)^\beta m_{c'}^\beta(f_h), \quad |\alpha| \le p, \tag{30}$$

where $\alpha! = \alpha_1!\alpha_2!\alpha_3!$ and $m_{c'}^\beta(f)$ is the moment of f_h, given by

$$m_{c'}^\beta(f) = \int_{S_{c'}} \frac{\partial^k}{\partial \nu_{\mathbf{x}}^k} (\mathbf{x} - \mathbf{x}_{c'})^\beta f_h(\mathbf{x}) dS_{\mathbf{x}}, \quad |\beta| \le p. \tag{31}$$

Equation (30) translates the moment of c' to the local expansion coefficients of the cluster c, and it is therefore called MtL translation. Since f_h is a linear combination of basis functions, we obtain from linearity that

$$m_{c'}^\beta(f) = \sum_{i \in c'} m_{c'}^\beta(\psi_i) f_i \tag{32}$$

where f_i are the coefficients of f_h with respect to the ψ_i-basis and the summation is taken over basis functions that whose support overlaps with c. Since we consider a Galerkin discretization we have to integrate the function $u_{c,c'}(\mathbf{x})$ against the test functions, to obtain the contribution g_i of the two clusters to the matrix vector product. Thus we get from (29)

$$g_i = \int_c \psi_i(\mathbf{x}) u_{c,c'}(\mathbf{x}) dS_{\mathbf{x}} = \sum_{|\alpha|\le p} \lambda_c^\alpha \int_c \frac{\partial^l}{\partial \nu_{\mathbf{x}}^l} (\mathbf{x}-\mathbf{x}_c)^\alpha \psi_i(\mathbf{x}) dS_{\mathbf{x}} = \sum_{|\alpha|\le p} \lambda_c^\alpha m_c^\alpha(\psi_i) \tag{33}$$

This operation converts expansion coefficients to potentials and is denoted as LtP translation.

To move moments and local expansion coefficients between levels, we also need the moment-to-moment (MtM) and local-to-local (LtL) translations. They can be derived easily from the multivariate binomial formula. We obtain

$$m_c^\alpha(f) = \sum_{c'\in\mathcal{K}(c)} \sum_{\beta\le\alpha} \binom{\alpha}{\beta} (\mathbf{x}_{c'} - \mathbf{x}_c)^{\alpha-\beta} m_{c'}^\beta(f), \tag{34}$$

and

$$\lambda_{c'}^\beta = \sum_{\substack{\alpha\le\beta \\ |\alpha|\le p}} \binom{\alpha}{\beta} (\mathbf{x}_{c'} - \mathbf{x}_c)^{\alpha-\beta} \lambda_c^\alpha, \quad |\beta| \le p, \tag{35}$$

where $c' \in \mathcal{K}(c)$.

We see that moments and expansion coefficients are computed by recurrence from the previous level. In the finest level the moments of the basis functions $m_c(\psi_i)$ can be either computed by numerical quadrature, or even analytically, because we consider flat panels and polynomial ansatz functions. We skip the details, as these formulas are straightforward application of the binomial formula.

In summary, the Cartesian FMM under the framework of boundary element method is described as the following.

1. *Nearfield Calculation.*
   for $c \in C_L$
      for $c' \in \mathcal{N}(c)$
         multiply matrix block of c and c' directly.
2. *Moment Calculation.*
   for $c \in C_L$
      Compute the moments $m_{c'}^\beta(f)$ in (32).
3. *Upward Pass.*
   for $l = L-1, \dots, l_{min}$
      for $c \in C_l$
         for $c' \in \mathcal{K}(c)$
            Compute the MtM translation (34)
4. *Interaction Phase.*
   for $l = L, \dots, l_{min}$
      for $\nu \in C_l$
         for $\nu' \in \mathcal{N}(c)$
            Compute the MtL translation (30)
5. *Downward Pass.*
   for $l = l_{min}, \dots, L-1$
      for $c \in C_l$
         for $c' \in \mathcal{K}(c)$
            Compute the LtL translation (35)
6. *Evaluation Phase.*
   for $c \in C_L$
      Compute the LtP translation (33)

In this algorithm l_{min} is the coarsest level that contains clusters with non-empty interaction lists.

Since the finest level contains a fixed number of triangles the number of levels grows logarithmically with N as the mesh is refined. With a geometric series argument, one can show that the total number of interaction lists in all levels is $O(N)$. If the translations in all levels are computed with the same order p then the complexity of all translations is $O(Np^4)$. If the variable order scheme (26) is used with fixed p_L as L and N are increased, then the complexity reduces to $O(N)$, see [33].

2.5. Cartesian treecode

The Cartesian treecode can be considered as a fast multipole method without the downward pass. The computational cost of treecode is order of $O(N \log N)$ as opposed to the $O(N)$ FMM. However, the constants in this complexity estimate are smaller, and we found it to be useful for the computation of the electrostatic solvation free energy E_{sol} (25), where the

source and evaluation points are different and zero or limited near field calculations are required. The direct computation of the solvation energy as interactions between N boundary elements and N_c atomic centers has $O(N_cN)$ complexity. This is shown in Fig. 2b, in which a charge located at $\mathbf{x}_n$ will interact with induced charges (ϕ_1 or $\frac{\partial\phi_1}{\partial\nu}$) located at the center of each panel. These interactions consist of near field particle-particle interaction by direction summation and far field particle-cluster interaction controlled by maximum acceptance criterion (MAC) as specified below. For simplicity, we write the involved calculations as

$$E_{\text{sol}} = \sum_{n=1}^{N_c} q_n V_n = \sum_{n=1}^{N_c} q_n \sum_{l=1}^{N} \int_{\tau_l} \frac{\partial^k}{\partial \nu_{\mathbf{x}}^k} G(\mathbf{x}_n, \mathbf{x}) f(\mathbf{x}) dS_{\mathbf{x}}, \tag{36}$$

where G is Coulomb or screened Coulomb potential kernel, $k \in \{0, 1\}$, q_n's are partial charges, and f is either ϕ_1 or $\partial\phi_1/\partial\nu$.

In our implementation, we use the same clustering scheme of Γ_N as in the FMM. Instead of using the interaction lists to calculate the far field interaction, the treecode use the following multipole acceptance criterion (MAC) to determine if the particle and the cluster are well separated or thus a far-field particle-cluster interaction will be considered. This is similar to the separation ratio in the FMM. The MAC is given as

$$\frac{r_c}{R} \le \theta, \tag{37}$$

where $r_c = \max_{\mathbf{x}_j \in c} |\mathbf{x}_j - \mathbf{x}_c|$ is the cluster radius, $R = |\mathbf{x}_n - \mathbf{x}_c|$ is the particle-cluster distance, and $\theta < 1$ is a user-specified parameter. If the criterion is not satisfied, the program checks the children of the cluster recursively until either the MAC is satisfied or the leaves (the finest level cluster) are reached at which direct summation is applied. Overall, the treecode evaluates the potentials (36) as a combination of particle-cluster interactions and direct summations. Thus, when $\mathbf{x}_n$ and c are well-separated, the potential can be evaluated as

$$\int_c \frac{\partial^k}{\partial \nu_{\mathbf{x}}^k} G(\mathbf{x}_n, \mathbf{x}) f(\mathbf{x}) dS_{\mathbf{x}} \approx \sum_{|\beta|=0}^{p} D^\beta G(\mathbf{x}_n, \mathbf{x}_c)(-1)^\beta m'^\beta_c(f), \tag{38}$$

where the moment $m'^\beta_c(f)$ is calculated by the same operator-MtM in FMM.

The treecode method therefore can be concluded as

1. *Moment Calculation.*
   for $c \in C_L$
      Compute the moments $m^\beta_{c'}(f)$ in (32).
2. *Upward Pass.*
   for $l = L-1, \ldots, l_{min}$
      for $c \in C_l$
         for $c' \in \mathcal{K}(c)$
            Compute the MtM translation (34)
3. *Interaction Phase*
   for $n = 1, \ldots, N_c$
      $E_n = 0$
      for $c \in C_0$
         addCluster(c,$\mathbf{x}_n$,E_n)

where addCluster(c,$\mathbf{x}_n$,E_n) as shown below is a routine that recurses from the coarse clusters to the finer clusters until the separation is sufficient to use the Taylor series approximation

if $\mathbf{x}_n$ and $\mathbf{x}_c$ satisfy the MAC for c

$$E_n \mathrel{+}= \sum_{|\beta|=0}^{p} D^\beta \Phi(\mathbf{x}_n, \mathbf{x}_c)(-1)^\beta m'^\beta_c(f)$$

else if $\mathcal{K}(c) \neq \emptyset$
   for $c' \in \mathcal{K}(c)$
      addCluster(c',$\mathbf{x}_n$,E_n)
else

$$E_n \mathrel{+}= \int_c \frac{\partial^\kappa}{\partial \nu_{\mathbf{x}}^\kappa} \Phi(\mathbf{x}_n, \mathbf{x}) f(x) dS_{\mathbf{x}}$$

Note that steps 1 and 2 are analogous to the steps the in FMM, hence the addition of the treecode to the FMM code requires little extra work.

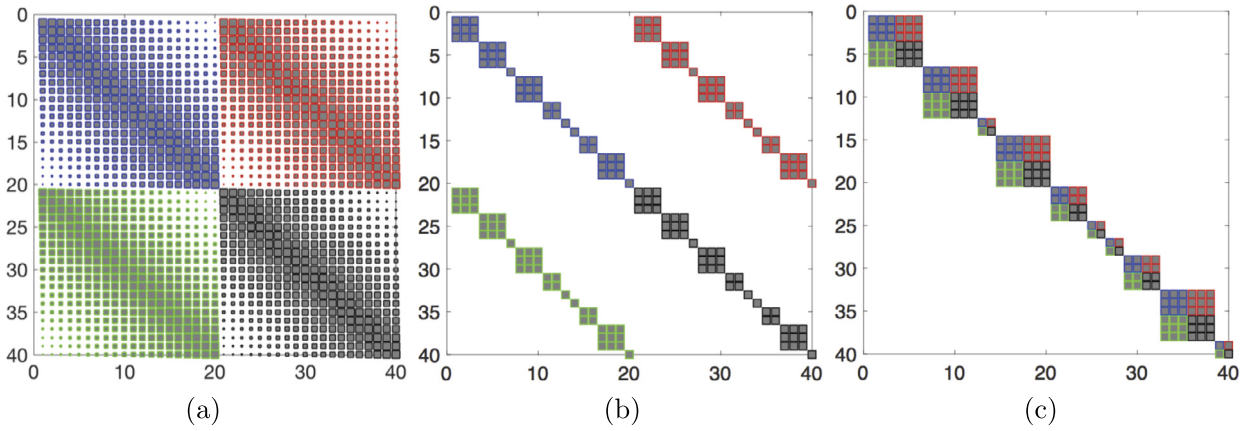

Fig. 3. A schematic illustration of the boundary element dense matrix A and its preconditioning matrix M: (a) matrix A for the case of $N = 20$ elements (the size of the matrix entry shows the strength of the interaction; the four different color-coded region relates to K_{1-4} in Eqs. (6a)-(6b)); (b) the "block diagonal block" preconditioning matrix M (assuming the cube at the designated level contains at most 3 panels; (c) the "block diagonal" preconditioning matrix M, which is a permuted matrix from M in (b) after switching the order of the unknowns.

2.6. Preconditioning

The results in our previous work [28,49] show that the PB boundary integral formulation in Eqs. (6a) and (6b) is well-conditioned thus will only require a small number of GMRES iteration if the triangulation quality is satisfied (e.g. nearly quasi-uniform). However, due to the complexity of the molecular surface, the triangulation unavoidably has a few triangles with defects (e.g. narrow triangles and tiny triangles) which deteriorate the condition number of the linear algebraic matrix, resulting in increased GMRES iteration number required to reach the desired convergence accuracy.

Recently, we designed a block-diagonal preconditioning scheme to improve the matrix condition for the treecode-accelerated boundary integral (TABI) Poisson-Boltzmann solver [41]. The essential idea for this preconditioning scheme is to use the short range interactions within the leaves of the tree to form the preconditioning matrix M. This preconditioning matrix M can be permuted into a block diagonal form thus $Mx = y$ can be solved by the efficient and accurate direct methods. In the current study of FAGBI solver, the same conditioning issue rises and it can be resolved by a similar but FMM structure adjusted and controlled preconditioning scheme.

The key idea is to find an approximating matrix M of A such that M is similar to A and the linear system $My = z$ is easy to solve. To this end, our choice for M is the matrix involving only direct sum interactions in cubes/clusters at a designated level (an optimal choice considering both cost and efficiency) as opposed to A, which involves all interactions.

The definition of M will be essentially similar to A in (22) except that the entries of M are zero if τ_i and τ_j are not on the same cube at a designated level of the tree, i.e.

$$M_{mn}(i,j) = \begin{cases} A_{mn}(i,j) & \text{if } \tau_i, \tau_j \text{ are on the same cube at a designated level of the tree} \\ 0 & \text{otherwise.} \end{cases} \tag{39}$$

Here we use Fig. 3 to illustrate how we design our preconditioning scheme and its advantage. Fig. 3(a) is the illustration of the dense boundary element matrix A for the discretized system (22) with 20 boundary elements. The four different colors represent the four kernels K_{1-4} related entries of the linear algebraic matrix A in Eq. (23). 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 on the same cube at a designated level, 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. Following the procedure detailed in [41], by rearranging the order of the unknowns, a block diagonal matrix M is achieved as illustrated in Fig. 3(c). 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 cube of the tree at a designated level. As shown in [41], the total cost of solving $My = z$ is essentially $O(N)$ thus is very efficient. Results for the preconditioning performance will be shown in the next section.

3. Results

Our numerical results are mostly produced on a PC desktop with an i5 7500 CPU and 16G Memory (3.1-3.3) and on a MAC Desktop with an i7 4.2 GHz Quad-Core Intel Core and 16GB memory (3.4), both using GNU Fortran compiler with compiling option "-O2". A few results for the long elapsed direct summation are obtained from the cluster available from SMU

Table 1
Discretization error from solving the PB equation on a Kirkwood sphere with a centered charge. Results include electrostatic solvation free energy $E_{\text{sol}}^{\text{ds}}$ with error $e_{\text{sol}}^{\text{ds}}$ and convergence rate $r_{\text{sol}}^{\text{ds}}$, and discretization error in surface potential e_{ϕ}^{ds}, normal derivative $e_{\partial_n\phi}^{\text{ds}}$ with their convergence rates r_{ϕ}^{ds} and $r_{\partial_n\phi}^{\text{ds}}$.

N[1]	h	$E_{\text{sol}}^{\text{ds}}$ (kcal/mol)	$e_{\text{sol}}^{\text{ds}}$ (%)	$r_{\text{sol}}^{\text{ds}}$	e_{ϕ}^{ds} (%)	r_{ϕ}^{ds}	$e_{\partial_n\phi}^{\text{ds}}$ (%)	$r_{\partial_n\phi}^{\text{ds}}$	Iters[2]
320	9.90	-8413.28	1.692	3.6	17.925	4.1	0.652	1.9	3
1280	4.95	-8328.18	0.663	2.6	4.419	4.1	0.212	3.1	3
5120	2.48	-8293.42	0.243	2.7	1.112	4.0	0.092	2.3	3
20,480	1.24	-8280.70	0.089	2.7	0.285	3.9	0.046	2.0	3
81,920	0.62	-8276.33	0.036	2.5	0.077	3.7	0.023	2.0	3
327,680	0.31	-8274.64	0.016	2.3	0.022	3.5	0.011	2.0	3
1,310,720	0.15	-8273.91	0.007	2.2	0.007	3.2	0.006	2.0	4
∞[3]		-8273.31							

[1] N is number of triangles in triangulation; h in Å is the average of largest edge length of all triangles; $h \approx O(N^{-1/2})$.
[2] Number of GMRES iterations.
[3] This row displays the exact electrostatic solvation energy $E_{\text{sol}}^{\text{ex}}$, which is known analytically [55].

Center for Research Computing (CRC), ManeFrame II (M2), with Intel Xeon Phi 7230 Processors, using OpenMPI compiler with compiling option "-O2". Note these direct summation results are needed for the evaluation of accuracy only. Before obtaining these results from HPC, we verified that low resolutions results computed on different machines are consistent. All protein structures are obtained from Protein Data Bank [50] and partial charges are assigned by CHARMM22 force field [51] using PDB2PQR software [52].

The physical quantity we computed in this manuscript is the electrostatic solvation free energy E_{sol} as defined in Eq. (36) 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 [52] 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=300K. From potential to electrostatic solvation free energy E_{sol}, only a multiplication of e_c is needed. For the PB model related parameters in the spherical case, the dielectric constant is $\varepsilon_1 = 1$ inside the sphere and $\varepsilon_2 = 40$ outside the sphere, and $\kappa = 0$. The reason for the choice of these values is to be consistent with some important previous work [25,28] for direct comparison. For protein cases, the dielectric constant is $\varepsilon_1 = 1$ in solute and $\varepsilon_2 = 80$ in solvent, and $\kappa = 0.1257$, which corresponds to $I_s = 0.15$M physiological saline (Note: $\kappa^2 = 8.430325455 I_s/\epsilon_2$). More details about PB equation related units conversion can be found in [53,54].

We solved the PB equation first on the Kirkwood sphere [55], where the analytic solution is available to validate the accuracy and efficiency of FAGBI solver, then on a typical protein 1a63 to demonstrate the overall performance on accuracy, efficiency, and memory usage, and the next on a series of 27 proteins to emphasize the preconditioning scheme, and the finally on 8 selected proteins with sizes ranging from 967 to 21,497 atoms for demonstrating the broad usage of the FAGBI solver and its $O(N)$ computational efficiency.

All numerical results reported here use constant basis functions for the discretization. For the test cases running on the Kirkwood sphere, we vary the order of the quadrature rule for the nearfield coefficients in (23) to study its relationship with accuracy. For test cases running on proteins, we use single point quadrature, which is also our recommended value for the practical usage of the FAGBI solver. The reported CPU time is the total time, which includes the small portion of time in generating the mesh by MSMS, the main portion of time in solving the PB equation, and the small portion of time in computing the solvation energy. Although machines with multi-cores are used, all tests are in serial with one core/thread being used. We will investigate the parallelization of FAGBI solver including its GPU implementation in future work.

3.1. Accuracy, CPU time, and memory tests on the Kirkwood sphere

Our first test case is the Kirkwood sphere of radius 50Å with an atomic charge $q = 50e_c$ at the center of the sphere. We provide three parts of this test case on the Kirkwood sphere, which show first the discretization error, then the impact of the quadrature orders toward the convergence of accuracy, and finally the comparison between using the Cartesian FMM and using direct sum in terms of error, CPU time, and memory usage.

3.1.1. Overall discretization error

We first solve the boundary integral PB equation on the Kirkwood sphere using the direct summation for matrix-vector product instead of using the FMM acceleration. The linear algebraic system is solved using GMRES iterative solver with L_2 relative tolerance $\tau = 10^{-6}$ and zero initial guess. The Galerkin method is applied to form the matrix combined with a single point Gauss quadrature. Cubature methods [47] are applied for treating the singularities arising from Galerkin discretization of boundary integral equations.

Table 1 shows the total discretization errors, which is related to triangulation, quadrature, and basis function. In this table, Column 1 is the number of triangles N for the sphere with the refinement of the mesh and Column 2 is the average of largest edge length of all triangles h. Note we have $h \approx O(N^{-1/2})$, which can be seen from the comparison of values in

Table 2
Discretization error of solvation free energy E^{ds}_{sol} for solving the same problem in case 1 using Gaussian quadrature orders of 2-4.

	Quad. Order 2			Quad. Order 3			Quad. Order 4		
N	E^{ds}_{sol}	r^{ds}_{sol}	Iters	E^{ds}_{sol}	r^{ds}_{sol}	Iters	E^{ds}_{sol}	r^{ds}_{sol}	Iters
320	-8378.82	3.7	2	-8369.13	3.9	2	-8369.21	3.9	2
1280	-8305.24	3.3	3	-8298.57	3.8	2	-8297.67	4.0	2
5120	-8284.01	3.0	3	-8280.12	3.7	3	-8279.40	3.9	3
20,480	-8277.27	2.7	3	-8275.21	3.6	3	-8374.81	4.0	3
81,920	-8274.94	2.4	3	-8273.89	3.3	3	-8373.68	4.1	3
327,680	-8274.03	2.3	3	-8273.51	3.0	3	-8373.40	4.0	3
1,310,720	-8273.64	2.2	3	-8273.38	2.7	3	-8273.33	4.7	3
∞	-8273.31			-8273.31			-8273.31		

the two columns. Since using N is more convenient to specify mesh refinement in our numerical simulation, we use it to quantify the mesh refinement for the rest of the paper.

Columns 3-4 show that the electrostatic solvation energy $E^{\text{ds}}_{\text{sol}}$ and its error $e^{\text{ds}}_{\text{sol}}$ compared with the true value in the last row of the table. The convergence rate $r^{\text{ds}}_{\text{sol}}$ defined as the ratio of the error is shown in column 5 with an $O(N^{-1/2})$ pattern. The relative L_∞ errors of surface potential ϕ, e^{ds}_{ϕ} and normal derivative $\partial_n\phi$, $e^{\text{ds}}_{\partial_n\phi}$, are shown in columns 6 and 8. The surface potential converges with a pattern of $O(N^{-1})$ as shown in column 7, which is faster than its normal derivative with a pattern of $O(N^{-1/2})$ as shown in column 9. We also can observe that the GMRES iterations shown in column 10 in all the tests are less than or equal to four, which verifies that the boundary integral formulation is well-posed.

Note a back-to-back comparison between Table 1 in this manuscript and Table 2 in our previous work [28] shows improvements in convergence of $E^{\text{ds}}_{\text{sol}}$, ϕ, and $\partial_n\phi$ for the present work. This is due to the Galerkin scheme with Duffy's trick and the Cubature method in treating the singularity as opposed to the collocation scheme with simply the removal of singular integral whenever it occurs on an element [28].

3.1.2. Quadrature error

In part 1, we noticed that the converge rate for $E^{\text{ds}}_{\text{sol}}$ is about $O(N^{-1/2})$ as the rate of $\partial_n\phi$, but is less than the $O(N^{-1})$ rate of ϕ. To investigate the possible reason, we study the influence of the quadrature rule the next.

We increase the order of the tensor product Gauss-Legendre rule to 2, 3 and 4 and test their effects on the discretization error of $E^{\text{ds}}_{\text{sol}}$ as shown in Table 2. Comparing with results in Table 1, using higher order quadratures improves both the convergence rate of $E^{\text{ds}}_{\text{sol}}$ and the required GMRES iterations. When the quadrature order is 4, the electrostatic solvation energy $E^{\text{ds}}_{\text{sol}}$ converges to the exact energy at the rate $O(N^{-1})$ approximately.

Increasing the quadrature order further will not significantly improve the convergence rate of $E^{\text{ds}}_{\text{sol}}$, because then the discretization error will be greater than the quadrature error. Since higher quadrature requires more computational cost, in practice, due to the large size of the protein solvation problem, we will use quadrature order 1 as it shows the optimal combination or accuracy and efficiency.

3.1.3. FMM

This part of the test studies the role of Cartesian FMM relating to the accuracy and efficiency of the algorithm. We applied the FMM to replace the direct-sum for accelerating the matrix-product calculation in GMRES. Here we use the first order quadrature rule for simplicity. We set $\eta = 0.8$, which is defined in (28) and adjust the number of levels L in the FMM algorithm for different N, such that with the increment of N, N_0, the maximum number of elements in a cluster at the finest level, is less than two times of its value associated with any previously used N. In other words, L will be increased by 1 each time N is quadrupled. Fig. 4 shows (a) the error in electrostatic solvation energy, (b) the CPU time, and (c) the memory usage versus the number of triangles N. Here the error is computed as compared with the exact value $E^{ex}_{\text{sol}} = -8273.31$. We provide results using fixed Taylor expansion order $p = 1, 3, 5, 7$ and adaptive order start from $p = 1$, and $p = 3$. Here the adaptive order represents the idea that expansion order should be adjusted to the level (e.g. higher expansion order at higher level) in order to match the discretization error [48]. In this figure, the solid blue line with square marks is results of direct summation with one point quadrature, which shows in (a) an $O(N^{-1/2})$ order of convergence in accuracy as observed in Table 1, an $O(N^2)$ CPU time in (b), and an $O(N)$ memory usage in (c).

As seen in Fig. 4(a), the use of FMM introduces truncation error in addition to the discretization error. Truncation errors are more significant than the discretization error when the order p is small and are less significant when p is large.

Furthermore, we observed that when expansion orders $p = 5, 7$ of the FMM are used, the errors are even smaller than those obtained with the direct sum. This is due to the fact that the error of the truncation error of Taylor approximation is smaller than the quadrature error of the far field coefficients in the direct sum.

As seen in Fig. 4(b), the use of FMM significantly reduces the CPU time, which shows a $O(N)$ pattern as opposed to the $O(N^2)$ pattern of the direct sum. Figs. 4(a-b) combined also justify the use of adaptive order. Adaptive order 1 and 3 use about the same amount of CPU time as regular order 1 and 3 but achieved significant improvements in accuracy. Meanwhile,

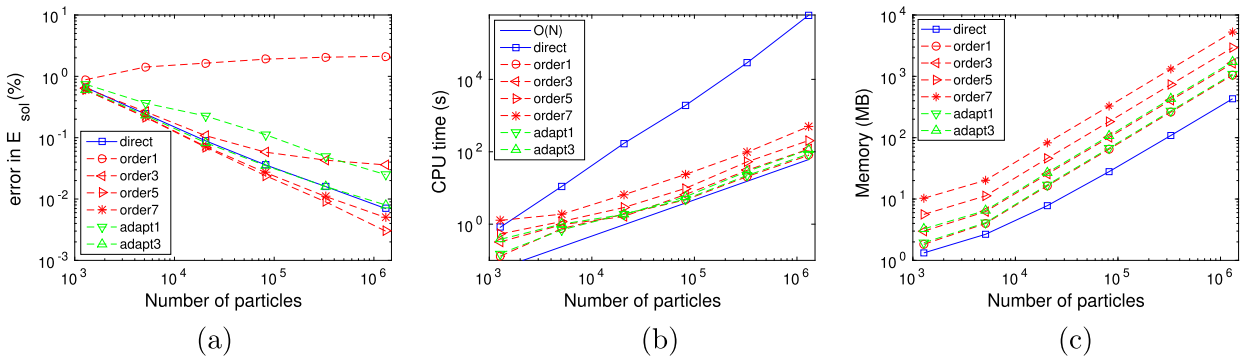

Fig. 4. Compute electrostatic solvation energy on a Kirkwood sphere as number of triangles/particles N in creases: (a) Error, (b) CPU time, and (c) Memory usage; discretization error e_{sol}^{ds} (solid line), Cartesian FMM approximation error e_{sol}^{cf} (dashed line); Taylor expansion order $p = 1, 3, 5, 7$ and adaptive Taylor order $p = 1, 3$.

Table 3
(protein 1A63). FAGBI results; PB equations; showing electrostatic solvation energy E_{sol}, error, CPU time, memory usage; columns show MSMS density (d in Å^{-2}), number of triangles N, E_{sol} values computed by direct sum (ds) and Cartesian FMM (cf), discretization error $e_{\text{sol}}^{\text{ds}}$, Cartesian FMM approximation error $e_{\text{sol}}^{\text{cf}}$ and their convergence rate $r_{\text{sol}}^{\text{ds}}$ and $r_{\text{sol}}^{\text{cf}}$; adaptive Taylor expansion order $p = 1$, separate rate $\eta = 0.8$.

d	N[a]	E_{sol} (kcal/mol)		Error (%)		Rate		CPU (s)		Mem. (MB)	
		ds	cf	$e_{\text{sol}}^{\text{ds}}$	$e_{\text{sol}}^{\text{cf}}$	$r_{\text{sol}}^{\text{ds}}$	$r_{\text{sol}}^{\text{cf}}$	ds	cf	ds	cf
1	20,227	-2755.05	-2756.82	16.12	16.20			632	6	18	30
2	30,321	-2498.20	-2499.20	5.30	5.34	5.5	5.4	1135	8	23	40
5	69,969	-2412.40	-2413.02	1.68	1.71	2.7	2.7	5912	17	66	111
10	132,133	-2383.09	-2382.50	0.45	0.42	4.1	4.4	36,530	37	92	165
20	264,927	-2375.21	-2376.52	0.11	0.17	3.9	2.6	149,651	69	249	423
40	536,781	-2371.52	-2372.77	0.04	0.01	2.9	7.5	618,879	141	359	654
	∞[b]	-2372.48	-2372.48								

[a] Number of elements in triangulation.
[b] This row shows the estimates of exact energy $E_{\text{sol}}^{\text{ex}}$ obtained by the parallel computing on high order quadrature method.

Fig. 4(c) shows that FMM use additional memory in trading of efficiency. However, the $O(N)$ pattern of memory usage has been well preserved at different orders with only an adjustment in a factor.

In summary, from Tables 1 and 2, we observed that the Galerkin discretization with piecewise constant basis functions can achieve $O(N^{-1/2})$ convergence rate with low quadrature order (e.g. 1 or 2) and can achieve $O(N^{-1})$ convergence rate with high quadrature order (e.g. 4). Applying FMM algorithm for acceleration significantly reduced the $O(N^2)$ CPU time to $O(N)$ while maintains desired accuracy and $O(N)$ memory usage. For later tests, we apply the adaptive FMM with starting order 1 and $\eta = 0.8$ as an optimal choice at the consideration of both efficiency and accuracy.

3.2. Accuracy, CPU time, and memory tests on protein 1a63

In this section, we use the FAGBI solver to compute the solvation energy for protein 1A63, which has 2065 atoms. In computation involving proteins, the molecular surface is triangulated by MSMS [42], with atom locations from the Protein Data Bank [50] and partial charges from the CHARMM22 force field [51]. MSMS has a user-specified density parameter d controlling the number of vertices per Å^2 in the triangulation. MSMS constructs an irregular triangulation which becomes smoother as d increases. The GMRES tolerance is $\tau = 10^{-4}$. These are representative parameter values chosen to ensure that the FMM approximation error and GMRES iteration error are smaller than the direct sum discretization error, and to keep efficient performance in CPU time and memory based on tests on spheres previously.

In Table 3, the first two columns give the MSMS density (d) and number of faces N in the triangulation. The next two columns give the electrostatic solvation energy E_{sol} computed by direct sum (ds) and Cartesian FMM (cf). We use a parallel version of direct sum to compute an estimate of the exact energy with high order quadrature methods. We computed the discretization errors $e_{\text{sol}}^{\text{ds}}$ and $e_{\text{sol}}^{\text{cf}}$ on the fifth and sixth columns, which shows convergence rate faster than $O(N^{-1})$ as observed for the geodesic grid triangulation of the Kirkwood sphere in Case 1. The faster convergence seen here is due to non-uniform adaptive treatment of MSMS triangulation [28].

A back-to-back comparison of results from direct sum (ds) and Cartesian FMM results (cf) in Error, Rate, CPU time, and Memory in Table 3 provides the following conclusions. (1) the adoption of FMM only slightly modifies the error and its convergence rate in accuracy, not even necessarily in a negative way; (2) Cartesian FMM dramatically reduces the $O(N^2)$ direct sum CPU time to $O(N)$. For example, the simulation with $d = 10\text{Å}^{-2}$ and $N = 132,133$ took $36,530$ s $\approx$ 10h by direct

Table 4
Convergence comparison using diagonal preconditioning (d) and block diagonal preconditioning (bd) on a set of 27 proteins; MSMS density $d = 10$.

Ind.	PDB	# of ele.	E_{sol} (kcal/mol)			# of it.		CPU time (s)		
			d	bd	diff. (%)	d	bd	d	bd	ratio
1	1ajj	40496	-1141.17	-1141.15	0.00	22	14	12.5	9.7	1.28
2	2erl	43214	-953.43	-953.42	0.00	15	10	9.2	7.8	1.18
3	1cbn	44367	-305.94	-305.94	0.00	12	11	7.4	8.3	0.88
4	1vii	47070	-906.11	-906.11	0.00	16	14	10.6	11.5	0.92
5	1fca	47461	-1206.46	-1206.48	0.00	16	11	10.2	8.8	1.16
6	1bbl	49071	-991.21	-991.22	0.00	19	13	13.3	11.2	1.18
7	2pde	50518	-829.49	-829.46	0.00	**75**	23	50.7	19.5	**2.60**
8	1sh1	51186	-756.64	-756.63	0.00	$\underline{100^{+}}$	21	70.7	18.2	**3.89**
9	1vjw	52536	-1242.55	-1242.56	0.00	11	10	8.2	9.3	0.87
10	1uxc	53602	-1145.38	-1145.38	0.00	20	13	14.7	11.9	1.23
11	1ptq	54256	-877.83	-877.84	0.00	16	13	11.9	12.2	0.97
12	1bor	54628	-857.28	-857.27	0.00	14	13	10.9	12.5	0.87
13	1fxd	54692	-3318.18	-3318.14	0.00	10	10	7.8	9.9	0.79
14	1r69	57646	-1094.86	-1094.86	0.00	13	12	10.6	12.6	0.84
15	1mbg	58473	-1357.32	-1357.33	0.00	18	13	14.8	13.6	1.09
16	1bpi	60600	-1309.61	-1310.02	0.03	18	12	16.2	14.5	1.11
17	1hpt	61164	-816.47	-817.34	0.11	15	13	12.8	14.0	0.92
18	451c	79202	-1031.74	-1031.91	0.02	27	20	30.3	28.8	1.05
19	1svr	88198	-1718.97	-1718.97	0.00	15	12	21.4	21.3	1.01
20	1frd	81792	-2868.29	-2867.32	0.00	14	12	18.1	17.2	1.05
21	1a2s	84527	-1925.23	-1925.24	0.00	20	17	26.4	24.8	1.06
22	1neq	89457	-1740.50	-1740.49	0.00	19	15	26.7	22.8	1.17
23	1a63	132133	-2382.50	-2382.50	0.00	21	16	41.3	36.8	1.12
24	1a7m	147121	-2171.13	-2172.12	0.00	**55**	21	111.2	51.4	**2.16**
25	2go0	111615	-1968.61	-1968.65	0.00	**44**	24	67.6	43.0	1.57
26	1uv0	128497	-2296.43	-2296.43	0.00	**73**	25	130.7	52.6	**2.48**
27	4mth	123737	-2479.62	-2479.61	0.00	**36**	18	64.3	37.0	1.74

sum and 37 s $\approx$ 1/2min by FMM; (3) Moreover, the memory usage shows that both the direct sum and FMM memory usage is $O(N)$. For the FMM, more memory is used for the moment and local coefficient storage but this only adds a pre-factor rather than increases the growth rate.

Note the data in Table 3 in this manuscript and Table 4 in our previous work [28] use the same protein 1a63 with only a slight modification of triangulation and referencing solvation energy. Thus it is reasonable to compare the performance between TABI solver and FAGBI solver. The FAGBI solver shows improvements in accuracy due to the Galerkin approach and the singularity removal treatment and enhancement in CPU due to the $O(N)$ Cartesian FMM. Both solvers use memory in $O(N)$ and the TABI solver uses less memory since no near field information is stored. Due to the simplicity of the algorithm, the TABI solver is relatively easier in parallelization, which is discussed and demonstrated in [56]. We will investigate the parallelization of FAGBI in future work.

3.3. Preconditioning tests on selected twenty-seven proteins

We the next provide testing results on a set of 27 proteins for the purpose of demonstrating the efficiency of the preconditioning scheme. Table 4 shows the convergence tests using diagonal preconditioning (d) and block diagonal preconditioning (bd) for a set of 27 proteins. After applying the block diagonal preconditioning scheme, the cases with slow convergence using diagonal preconditioning have been well resolved. In this table, the first column is the protein index, followed by the PDB ID in the second column, and the number of elements in the third column generated by MSMS with density $d = 10$. Columns 4 and 5 are the solvation energy of the proteins applying both preconditioning schemes, and column 6 is the relative difference between both methods, which shows no significant difference. A significant reduction of number of iterations using block diagonal preconditioning (bd) is shown in column 8 compared with results in column 7 using diagonal preconditioning (d). One can see that the worse the diagonal preconditioning result is, the larger improvements block diagonal preconditioning can achieve. For example, proteins 2pde, 1sh1, 1a7m, 2go0, 1uv0 and 4mth first use 75, 100^{+}(maximum number of iteration is reached), 55, 44, 73, and 36 iterations for diagonal preconditioning as highlighted in column 7, but only use 23, 21, 21, 24, 25, and 18 iterations for block diagonal preconditioning. The CPU time comparison in columns 9 and 10, as well as their ratio in column 11, further confirms the results in columns 7 and 8 as CPU time is related to the number of iterations. The ratio of CPU reduction for some proteins is more than 2 times as highlighted in the last column. We plot the results of columns 7,8,9 and 10 in Fig. 5 which 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 the block diagonal preconditioning does not impair the originally well-conditioned cases but significantly improve the slow convergence cases, which suggests that we can uniformly use block diagonal preconditioning in replace of the original

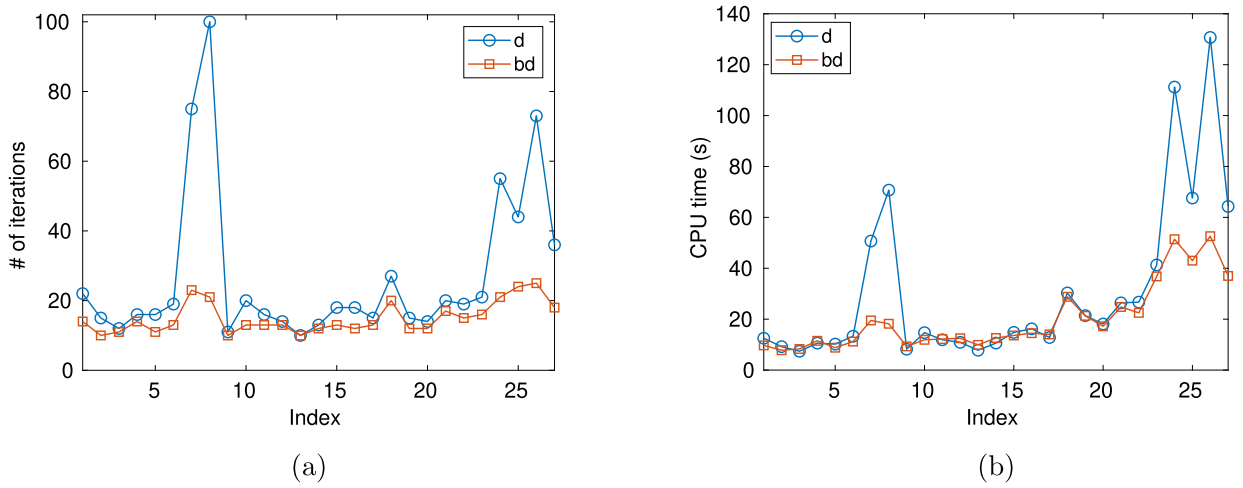

Fig. 5. Convergence comparison using diagonal preconditioning and block diagonal preconditioning: (a) number of iterations; (b) CPU time (s).

diagonal preconditioning. Figs. 5(a) and 5(b) shows a similar pattern as CPU time and the number of iterations are highly correlated.

3.4. Accuracy and efficiency tests on selected eight proteins

Finally, we test the performance of our FAGBI solver on 8 selected proteins with even larger size range. These proteins and their number of atoms are listed below.

PDB	4mn3	3dxg	3kgp	3udh	3f3c	2x8z	3ebp	2zx6
# of Atoms	967	1856	3533	6112	8183	9560	13127	21497

For these proteins, the number of atoms ranges from 967 to 21497. For each protein, we also refine the mesh using MSMS density parameter d from $d = 1$ to $d = 16$ by doubling d at each step, resulting in number of elements ranging from 10,966 to 1,422,568.

The results are reported in Table 5 and Fig. 6. The figure shows the pattern of convergence for solvation energy and the pattern of CPU time growth when the meshes are refined, while the table provides the detailed data. Our explanation focuses on the figure. Fig. 6(a) plots the solvation energy E_{sol} against the number of triangular elements N taken from the bottom part of Table 5. It is clear that for each tested protein, E_{sol} converges when the mesh is refined as the difference for E_{sol} received at different N narrows with the increment of N. Fig. 6(b) plots the total CPU time against the number of triangular elements N, taken from the top part of Table 5. We can see for each protein the total CPU time is consistent with $O(N)$ pattern. Note in Cartesian FMM, the depth or level of the tree should be increased when the mesh is repeatedly refined, otherwise there will be higher and higher portion of direct-sum computation with the increment of N, which deteriorates the $O(N)$ computational cost pattern. The general rule is that the depth or level of tree should be increased by 1 every time the mesh is refined by approximately four times. By applying this rule, we increase the tree depth by one at $d = 4$ and $d = 16$ to maintain the $O(N)$ CPU growth pattern. The tree depth increment also explains the pattern observed on the graph such that at $d = 4$ and $d = 16$, the CPU time used is lower than that from the $O(N)$ pattern. It is a piecewise linear pattern with each new piece lower than the linear extension of the previous piece.

It is worth mentioning that we can see from Table 5, the number of iteration n_i is stable for each protein at various meshes, owing to the well-posed integral form and preconditioning scheme which alleviates ill-conditioning possibly caused by triangulation quality. The strength of preconditioning, which is represented as the band width of the block diagonal preconditioning matrix, is determined by N_0, the maximum number of elements in a cluster at the finest level. The refinement of mesh causes the increment of N, which increases N_0 while the increment of tree depth causes the reduction of N_0. The variation of N_0 is the major reason for the variation of n_i as observed in some test cases shown in the table.

Furthermore, the CPU time is determined by number of atoms (or charges) N_c, number of elements N, and number of GMRES iterations n_i combined in the form of $O(n_i N) + O(N_c \log N)$. Assuming N increases with N_c in $O(N_c)$, in order to receive a solver with desired performance $O(N_c)$, the remedy is to use smaller N thus the $\log N << N_c$ at the price of sacrificing certain level of accuracy.

As suggested by the reviewers, we provide a back-to-back comparison for the performance between the TABI solver [28] and the present FAGBI solver in terms of solvation energy, number of GMRES iteration, memory usage, and CPU time on protein 2x8z at different MSMS densities. The results produced using the same machine as specified earlier are reported

Table 5
Solvation energy E_{sol} and total CPU time t from solving PB equation on eight selected proteins; MSMS density $d = 1, 2, 4, 8, 16$; Here N is number of triangular elements and n_i is the number of iterations using GMRES.

PDB	$d=1$			$d=2$			$d=4$			$d=8$			$d=16$		
	N	t	n_i	N	t	n_i	N	t	n_i	N	t	n_i	N	t	n_i
4mn3	10966	3	11	16428	4	10	30582	7	13	58454	15	13	122098	27	14
3dxg	16688	5	11	24782	10	14	46188	16	19	87874	29	14	183952	66	22
3kgp	26836	14	15	39804	27	17	74032	32	19	140123	98	23	291794	134	23
3udh	45796	57	27	68224	91	22	125606	83	20	237220	217	20	492548	311	22
3f3c	58092	61	17	85198	150	24	152260	109	18	281640	296	20	583052	510	28
2x8z	67784	110	28	100574	198	26	182062	170	24	344574	415	22	711296	641	27
3ebp	82850	113	19	122434	222	20	221834	225	22	417326	582	22	863321	808	23
2zx6	135972	1110	52	200291	1385	30	364580	765	22	688770	2164	20	1422568	2536	23
	N	E_{sol}		N	E_{sol}		N	E_{sol}		N	E_{sol}		N	E_{sol}	
4mn3	10966	-1878.78		16428	-1580.66		30582	-1501.26		58454	-1477.47		122098	-1467.65	
3dxg	16688	-2227.16		24782	-1867.18		46188	-1765.18		87874	-1736.33		183952	-1722.58	
3kgp	26836	-2926.11		39804	-2464.15		74032	-2305.60		140123	-2258.84		291794	-2237.79	
3udh	45796	-5551.95		68224	-4773.15		125606	-4514.52		237220	-4435.15		492548	-4411.82	
3f3c	58092	-4933.45		85198	-4307.33		152260	-4112.13		281640	-4057.60		583052	-4036.67	
2x8z	67784	-12898.83		100574	-11588.09		182062	-11191.14		344574	-11063.83		711296	-11025.48	
3ebp	82850	-8077.99		122434	-6716.17		221834	-6305.20		417326	-6177.53		863321	-6133.47	
2zx6	135972	-11566.84		200291	-9824.83		364580	-9161.03		688770	-8993.91		1422568	-8931.67	

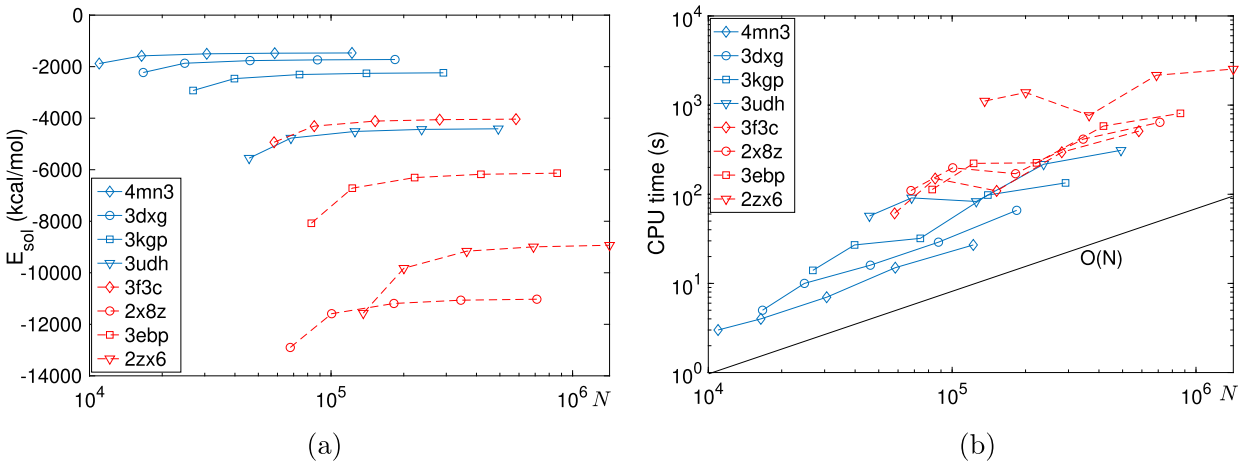

Fig. 6. Solvation energy E_{sol} and total CPU time from solving PB equation on eight selected proteins; N is the number of triangular elements.

Table 6
Solvation energy E_{sol}, number of GMRES iteration n_i, total memory usage m in kilobyte, and total CPU time t from solving PB equation on protein 2x8z at MSMS density $d = 1, 2, 4, 8, 16$ using both the TABI solver and the FAGBI solver.

	$d=1, N=67784$		$d=2, N=100574$		$d=4, N=182062$		$d=8, N=344574$		$d=16, N=711296$	
	E_{sol}	n_i	E_{sol}	n_i	E_{sol}	n_i	E_{sol}	n_i	E_{sol}	n_i
FAGBI	-12898.83	28	-11588.09	26	-11191.14	24	-11063.83	22	-11025.48	27
TABI	-13212.95	21	-11716.28	20	-11307.77	19	-11140.77	18	-11068.99	18
	m	t	m	t	m	t	m	t	m	t
FAGBI	78976	110	108820	198	203700	170	338580	415	801768	641
TABI	115432	225	158312	335	242600	606	413472	1256	835256	2742

in Table 6. From this table, we can see the solvation energies computed with both solvers converge to each other with the increase of the density by showing less and less difference. Both solvers use similar amount of memory with $O(N)$ pattern. Although the FAGBI solver uses a few more GMRES iteration than the TABI solver, which is the combined effects from resulting linear algebra matrix and the adopted preconditioner, its efficiency in $O(N)$ surpass the $O(N \log N)$ efficiency of TABI solver more and more significantly with the increment of the size of the problem.

Also as suggested by the reviewers, we add an example on the protein 1a63showing that the triangulation quality affects the GMRES convergence and the preconditioner designed improves the convergence. We compare the triangulation of the two software MSMS [42] and Nanoshaper [44], which produce similar number of triangles (20240 for MSMS and 20388 for

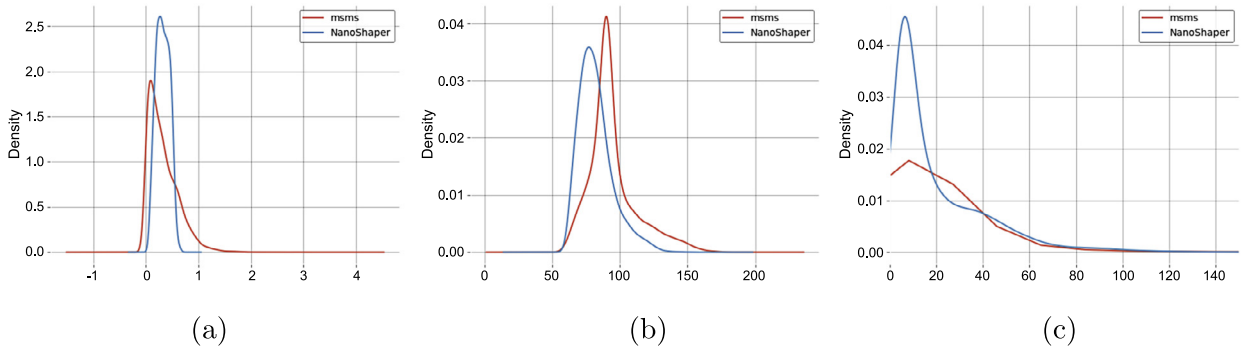

Fig. 7. Histograms of triangulation quality statistics for protein 1a63 whose molecular surface triangulations are generated by NanoShaper and MSMS: (a) triangle areas in $Å^2$, (b) maximum interior angles in degree, and (c) aspect ratios.

Nanoshaper). Without the preconditioner, the TABI solver uses 23 and 14 number of iterations with MSMS and Nanoshaper triangulations respectively. With the preconditioner, the TABI solver uses 11 iterations with both MSMS and NanoShaper triangulations. Fig. 7 provides the histograms of triangulation of both MSMS and NanoShaper in terms of triangle areas, maximum interior angles, and aspect ratios (outliers are omitted for better illustration). These histograms show that the triangulation quality from NanoShaper is better than that from MSMS thus the solver takes less number of GMRES iterations using NanoShaper triangulation than using MSMS triangulation.

4. Conclusion

In this paper, we report recent work in developing an FMM accelerated Galerkin boundary integral (FAGBI) method for solving the Poisson-Boltzmann equation. The solver has combined advantages in accuracy, efficiency, and memory as it applies a well-posed boundary integral formulation to circumvent many numerical difficulties and uses an $O(N)$ Cartesian FMM to accelerate the GMRES iterative solver. Special treatments such as adaptive FMM order, block diagonal preconditioning, Galerkin discretization, and Duffy's transformation are combined to improve the performance, which is validated on benchmark Kirkwood's sphere and a series of testing proteins. With its attractive $O(N^{-1})$ convergence rate in accuracy, $O(N)$ CPU run time, and $O(N)$ memory usage, the FAGBI solver and its broad usage can contribute significantly to the greater computational biophysics/biochemistry community as a powerful tool for the study of electrostatics of solvated biomolecules. We have released the code for the FAGBI solver on GitHub (https://github.com/gengwh/CFMM-PB) and we welcome interested users to apply the code for solving the Poisson-Boltzmann model. Feel free to contact us for questions and technique support.

CRediT authorship contribution statement

Jiahui Chen: Methodology, Software, Validation, Writing – original draft. **Johannes Tausch:** Methodology, Software, Writing – review & editing. **Weihua Geng:** Methodology, Project administration, Supervision, Writing – original draft.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Weihua Geng reports financial support was provided by National Science Foundation. Jiahui Chen reports financial support was provided by National Science Foundation. Johannes Tausch reports financial support was provided by National Science Foundation.

Data availability

Data will be made available on request.

Acknowledgements

The work of W.G. and J.C. was supported by NSF grant DMS-1819193, DMS-2110869, SMU new faculty startup fund and SMU center for research computing (CRC). The work of J.T. is in part funded by NSF grant DMS-1720431.

References

[1] 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.
[2] 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.
[3] 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.
[4] 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.
[5] J. Chen, J. Hu, Y. Xu, R. Krasny, W. Geng, Computing protein pKas using the TABI Poisson-Boltzmann solver, J. Comput. Biophys. Chem. 20 (2) (2021) 175–187.
[6] Y.C. Zhou, B. Lu, A.A. Gorfe, Continuum electromechanical modeling of protein-membrane interactions, Phys. Rev. E 82 (Oct 2010) 041923.
[7] 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.
[8] 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.
[9] N. Unwin, Refined structure of the nicotinic acetylcholine receptor at 4Å resolution, J. Mol. Biol. 346 (4) (2005) 967–989.
[10] N.A. Baker, Poisson-Boltzmann methods for biomolecular electrostatics, Methods Enzymol. 383 (2004) 94–118.
[11] 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.
[12] B. Honig, A. Nicholls, Classical electrostatics in biology and chemistry, Science 268 (5214) (1995) 1144–1149.
[13] M. Mirzadeh, M. Theillard, A. Helgadöttir, D. Boy, F. Gibou, An adaptive, finite difference solver for the nonlinear Poisson-Boltzmann equation with applications to biomolecular computations, Commun. Comput. Phys. 13 (1) (2013) 150–173.
[14] R. Luo, L. David, M.K. Gilson, Accelerated Poisson-Boltzmann calculations for static and dynamic systems, J. Comput. Chem. 23 (13) (2002) 1244–1253.
[15] 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.
[16] Z. Qiao, Z. Li, T. Tang, Finite difference scheme for solving the nonlinear Poisson-Boltzmann equation modeling charged spheres, J. Comput. Math. 24 (2006) 04.
[17] S. Yu, W. Geng, G.W. Wei, Treatment of geometric singularities in implicit solvent models, J. Chem. Phys. 126 (2007) 244108.
[18] R. Egan, F. Gibou, Geometric discretization of the multidimensional Dirac delta distribution – application to the Poisson equation with singular source terms, J. Comput. Phys. 346 (2017) 71–90.
[19] W. Geng, S. Yu, G.W. Wei, Treatment of charge singularities in implicit solvent models, J. Chem. Phys. 127 (2007) 114106.
[20] Q. Cai, J. Wang, H.-K. Zhao, R. Luo, On removal of charge singularity in Poisson-Boltzmann equation, J. Chem. Phys. 130 (14) (2009).
[21] W. Geng, S. Zhao, A two-component matched interface and boundary (MIB) regularization for charge singularity in implicit solvation, J. Comput. Phys. 351 (2017) 25–39.
[22] A. Lee, W. Geng, S. Zhao, Regularization methods for the Poisson-Boltzmann equation: comparison and accuracy recovery, J. Comput. Phys. 426 (2020) 109958.
[23] A. Juffer, B. E., B. E., 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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 (2013) 107–128.
[28] W. Geng, R. Krasny, A treecode-accelerated boundary integral Poisson-Boltzmann solver for electrostatics of solvated biomolecules, J. Comput. Phys. 247 (2013) 62–78.
[29] 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.
[30] E. Cancès, Y. Maday, B. Stamm, Domain decomposition for implicit solvation models, J. Chem. Phys. 139 (5) (2013) 054111.
[31] C. Quan, B. Stamm, Y. Maday, A domain decomposition method for the Poisson–Boltzmann solvation models, SIAM J. Sci. Comput. 41 (2) (2019) B320–B350.
[32] 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.
[33] J. Tausch, The variable order fast multipole method for boundary integral equations of the second kind, Computing 72 (3) (2004) 267–291.
[34] J. Barnes, P. Hut, A hierarchical O(N log N) force-calculation algorithm, Nature 324 (1986) 446–449.
[35] P. Li, H. Johnston, R. Krasny, A Cartesian treecode for screened Coulomb interactions, J. Comput. Phys. 228 (10) (2009) 3858–3868.
[36] L. Wang, R. Krasny, S. Tlupova, A kernel-independent treecode based on barycentric Lagrange interpolation, Commun. Comput. Phys. 28 (4) (2020) 1415–1436.
[37] J. Chen, W. Geng, D. Reynolds, Cyclically paralleled treecode for fast computing electrostatic interactions on molecular surfaces, Comput. Phys. Commun. 260 (2021) 107742.
[38] 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.
[39] 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.
[40] M.G. Duffy, Quadrature over a pyramid or cube of integrands with a singularity at a vertex, SIAM J. Numer. Anal. 19 (6) (1982) 1260–1262.
[41] J. Chen, W. Geng, On preconditioning the treecode-accelerated boundary integral (TABI) Poisson-Boltzmann solver, J. Comput. Phys. 373 (2018) 750–762.
[42] M.F. Sanner, A.J. Olson, J.C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.
[43] 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.

[44] S. Decherchi, W. Rocchia, A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale, PLoS ONE 8 (2013) 1–15.
[45] R. Egan, F. Gibou, Fast and scalable algorithms for constructing Solvent-Excluded Surfaces of large biomolecules, J. Comput. Phys. 374 (2018) 91–120.
[46] S. Rjasanow, O. Steinbach, The Fast Solution of Boundary Integral Equations, Springer, 2006.
[47] S.A. Sauter, Cubature techniques for 3-D Galerkin BEM, in: Boundary Elements: Implementation and Analysis of Advanced Algorithms, vol. 50, 1996, pp. 29–44.
[48] J. Tausch, The fast multipole method for arbitrary Green's functions, Contemp. Math. 329 (2003) 307–314.
[49] W. Geng, F. Jacob, A GPU-accelerated direct-sum boundary integral Poisson-Boltzmann solver, Comput. Phys. Commun. 184 (6) (2013) 1490–1496.
[50] http://www.rcsb.org/pdb/home/home.do.
[51] 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.
[52] 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).
[53] M.J. Holst, The Poisson-Boltzmann Equation: Analysis and Multilevel Numerical Solution, PhD thesis, UIUC, 1994.
[54] D. Xie, New solution decomposition and minimization schemes for Poisson-Boltzmann equation in calculation of biomolecular electrostatics, J. Comput. Phys. 275 (2014) 294–309.
[55] J.G. Kirkwood, Theory of solution of molecules containing widely separated charges with special application to zwitterions, J. Comput. Phys. 7 (1934) 351–361.
[56] L. Wilson, W. Geng, R. Krasny, TABI-PB 2.0: an improved version of the treecode-accelerated boundary integral Poisson-Boltzmann solver, J. Phys. Chem. B 126 (2022) 7104–7113.