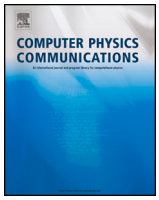
Computer Physics Communications 299 (2024) 109125

Contents lists available at ScienceDirect

Computer Physics Communications

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



Computational Physics

Optimized parallelization of boundary integral Poisson-Boltzmann solvers ☆

Xin Yang, Elyssa Sliheet, Reece Iriye, Daniel Reynolds, Weihua Geng *

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

ARTICLE INFO

Keywords:
Poisson-Boltzmann
Boundary integral
Treecode
MPI
GPU
COVID-19

ABSTRACT

The Poisson-Boltzmann (PB) model governs the electrostatics of solvated biomolecules, i.e., potential, field, energy, and force. These quantities can provide useful information about protein properties, functions, and dynamics. By considering the advantages of current algorithms and computer hardware, we focus on the parallelization of the treecode-accelerated boundary integral (TABI) PB solver using the Message Passing Interface (MPI) on CPUs and the direct-sum boundary integral (DSBI) PB solver using KOKKOS on GPUs. We provide optimization guidance for users when the DSBI solver on GPU or the TABI solver with MPI on CPU should be used depending on the size of the problem. Specifically, when the number of unknowns is smaller than a predetermined threshold, the GPU-accelerated DSBI solver converges rapidly thus has the potential to perform PB model-based molecular dynamics or Monte Carlo simulation. As practical applications, our parallelized boundary integral PB solvers are used to solve electrostatics on selected proteins that play significant roles in the spread, treatment, and prevention of COVID-19 virus diseases. For each selected protein, the simulation produces the electrostatic solvation energy as a global measurement and electrostatic surface potential for local details.

1. Introduction

Since the invention of X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy, the structures of biomolecules such as proteins and nucleic acids has become largely available to computational bioscientists. For example, as of October 13, 2023, there are 210,554 structures stored in the protein data bank (PDB) [1]. These available structures make the computational simulation of biomolecules possible, which tremendously promotes the advancement of structural biology research. At the molecular level, where length is normally measured in angstroms (Å), the structures, properties, functions, and dynamics of proteins are critically determined by the electrostatic interactions between proteins, ligands, and their solvent environment. The fundamental physical theory characterizing these electrostatics interactions is Gauss's law, the differential form of which leads to the Poisson-Boltzmann (PB) model that incorporates quantities such as media permittivity, protein charge distribution, electrolytes distribution in solvent, temperature, etc. With the assistance of numerical algorithms and computational tools, the PB model has broad applications in biomolecular simulations, including protein structure [12], protein-protein interaction [14,26], chromatin packing [3], protein pKa values, protein-membrane interactions, [7,48], binding energy [34,45], solvation free energy [39,43], ion channel profiling [40], etc.

Solving the PB model accurately and efficiently is challenging due to a variety of factors, such as the large problem dimension, the complex geometry of the protein, the jump of dielectric constants across media interface, and the singular charge representation. In the past 15 years, we have been working on developing fast and accurate PB solvers as useful tools for theoretical and computational bio-scientists [11,17,19–22]. Among the many attempted approaches, our favorite choices are the boundary integral PB solvers [19,20], which potentially have the best combination of efficiency, accuracy, memory usage, and parallelization. This manuscript will focus on the parallelization development of boundary integral PB solvers.

Boundary integral PB solvers are amenable to parallel computing because solving the boundary integral PB model is essentially similar to computing the pairwise interactions between the charges or induced charges located at the boundary elements, which is in fact an n-body problem. The parallelization of these pairwise interactions with $O(n^2)$ computational cost is straight-forward. When fast algorithms, such as treecode [30] or fast multipole method (FMM) [23] are involved, extra work is needed to ensure the parallel efficiency [9,47]. This article

☆ The review of this paper was arranged by Prof. W. Jong.

* Corresponding author.
E-mail addresses: xiny@smu.edu (X. Yang), esliheet@smu.edu (E. Sliheet), ririye@smu.edu (R. Iriye), reynolds@smu.edu (D. Reynolds), wgeng@smu.edu (W. Geng).

https://doi.org/10.1016/j.cpc.2024.109125
Received 16 December 2023; Received in revised form 3 February 2024; Accepted 6 February 2024
Available online 12 February 2024

emphasizes the comparison for parallelization between treecode and direct-sum when different computing hardware are available. With MPI implementation using multicore CPUs, for moderate numbers of particles we can build the tree on every CPU/task thus all particle-tree interactions can be concurrently done with high parallel efficiency [9,20]. When the number of particles are huge thus replicating the tree on each task is not possible, domain decomposition approach can be used instead [9,37].

In recent years, Graphic Processing Units (GPUs) have revolutionized computation-intensive tasks, including high performance computing, call center, autopilot, artificial intelligence, etc. One GPU card can consist of hundreds to thousands of cores and can rapidly implement large quantities of tasks, particularly *single instruction multiple data* (SIMD) tasks that benefit from large-scale concurrency. The n-body problem in which all particles concurrently interact with all other particles is thus very suitable for GPU computation. Calculation of n-body interactions using direct-sum on GPUs can be traced back to the first decade of the 21st century [16,35]. Later, when fast algorithms, such as treecode, were designed for the n-body problem, their GPU implementation is also possible [4,6,24]. However, these complex algorithms are challenging to implement on GPUs, thereby reducing the achievable level of parallel speedup. For example, as shown in [5], when more than 100k particles are involved in an n-body interaction, an NVIDIA GeForce 8800GTX GPU outperforms an Intel Xeon CPU running at 3.4 GHz by a factor of about 100 using direct summation as opposed to a factor of about 10 using Barnes-Hut treecode [2].

Our argument is that when a fast algorithm is available to replace the direct method e.g. treecode vs. direct summation, there is a break-even number n_b such that the fast algorithm should only be applied when the problem size exceeds this threshold. For example, for an n-body problem, consider the $c_1 n \log n$ treecode algorithm against the $c_2 n^2$ direct sum, where c_1 and c_2 are some constants calculated from the algorithms, the break-even number n_b is the n that satisfies the equation $c_1 n \log n = c_2 n^2$. Due to the algorithmic simplicity of direct summation, the break-even number n_b becomes considerately larger on GPUs in comparison to CPUs. Thus when the efficiency of the implementation is of critical importance, such as when these are used repeatedly within molecular dynamics or Monte Carlo simulations, we should use direct summation on GPU when the problem size is smaller than n_b. As we will show in the section of numerical results, n_b for the current GPU/CPU hardware conditions used for this project is about 250,000. In practice, molecular surface with 250,000 triangular elements can readily represent a large group of proteins with less than 2000-3000 atoms. For larger problems, we can use the MPI-based parallel treecode [9] or use the domain decomposition approach [37]. In short, every method has its own advantageous size of problems. We also note that the previous studies that have utilized a mix of hardware and methods, including the elegant work using MPI and GPUs [42]. We additionally point out the study by Wilson et al. [46] wherein an Orthogonal Recursive Bisection (ORB) tree based domain decomposition [37] is applied to allow each MPI task to only contain a local essential tree. The majority of the computation is then transferred to the GPU available on the node. This work is efficient for large problems but requires additional hardware support such that each node should have compatible multiple CPUs and a GPU card.

In this work we focus on two approaches for the parallelization of boundary integral PB solvers. One is the parallelization of the treecode-accelerated boundary integral (TABI) solver using MPI which builds an identical tree on each task/CPU. Its parallelization occurs at four stages of the TABI solver: source term computation, treecode for matrix-vector product, preconditioning, and energy computation. Among these stages, we apply the schemes developed for n-body parallelization [9] to a more complicated boundary integral PB problem, and develop MPI-based parallelization for the preconditioning scheme designed for boundary integral solvers [8]. Our second parallelization approach focuses on GPU-based parallelization of the direct-sum boundary integral (DSBI) solver, which concurrently computes the source term, matrix-vector product, and energy computation. We note that we did not use the preconditioning scheme in this approach since its GPU implementation is complicated and inefficient. As we will explain in the section of theories and algorithms and show in the section of numerical results, for some proteins whose surface triangulations contain a few poor-quality triangles, without using the preconditioning schemes will take longer time for the GMRES [36] iterative solver to converge, which is a limitation we are still working to improve.

The rest of the paper is organized as follows. In Section 2, we introduce the theories and algorithms involving the PB model, the boundary integral method, the treecode algorithm, the GMRES preconditioning scheme, the GPU and MPI parallelization, and biological information on the COVID-19 related proteins. Following that, we provide numerical results and related discussion in Section 3. This paper ends with short sections on software dissemination and concluding remarks.

2. Theories and algorithms

In this section, we will first briefly describe the PB model. Then we present a brief introduction to the treecode-accelerated boundary integral (TABI) PB solver. After that, we move to the parallelization schemes, which include a MPI based TABI solver and a GPU-accelerated direct-sum boundary integral (DSBI) Solver. This section ends with the introduction of the proteins that are involved in COVID-19 diseases, and are the target for our simulations.

2.1. The PB model

The PB model is illustrated in Fig. 1(a). Consider a domain Ω in $\mathbb{R}^3$ divided by the molecule surface Γ into the molecule domain Ω^- with dielectric constant ϵ^- and the solvent domain Ω^+ with dielectric constant ϵ^+, i.e., $\Omega = \Omega^- \bigcup \Omega^+$. Note that the surface Γ here is a 2-d illustration of a 3-d molecular surface of the protein 6yi3 whose triangulated molecular surface is given in Fig. 1(b). This triangulated surface provides the discretized space for the boundary integral method that we adopt. Note that all the proteins used in this article are from the Protein Data Bank (PDB) [1]. We refer to the proteins using their 4-digit PDB ID numbers, and detailed information on each can be found on the PDB website. Protein 6yi3 is one of the COVID-19 proteins of interest in this study, which will be described in the following sections.

Charges in Ω^- illustrated as circled "+" and "-" signs are partial charges assigned to the centers of atoms by using the force field, while charges in Ω^+ illustrated as "+" and "-" signs are mobile ions described by the Boltzmann distribution. Assuming equal amounts of positively and negatively charged electrolytes, then according to Gauss's law the charge distribution for $\mathbf{r} \in \Omega^+$

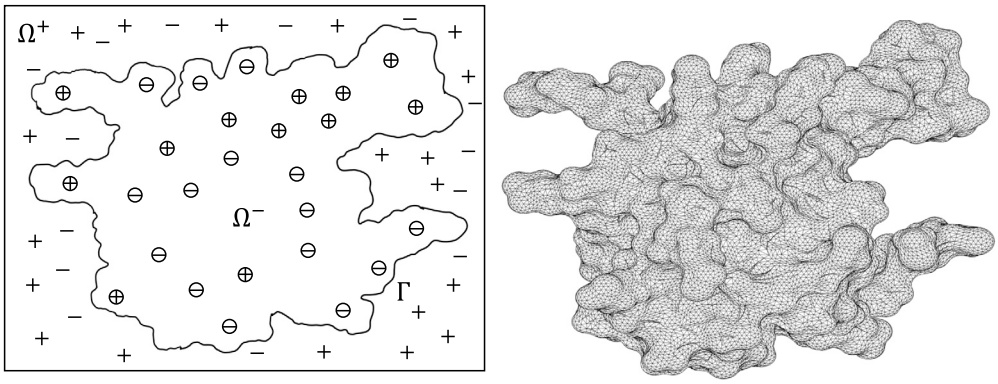
$$-\nabla \cdot (\epsilon(\mathbf{r})\nabla\phi(\mathbf{r})) + \bar{\kappa}^2(\mathbf{r})\sinh(\phi(\mathbf{r})) = \rho(\mathbf{r}), \tag{1}$$

subject to interface continuity for the potential ϕ and flux density $\epsilon\phi_{\mathbf{n}}$,

$$[\phi]_\Gamma = 0 \text{ and } [\epsilon\phi_\nu]_\Gamma = 0. \tag{2}$$

Here $\nu = (n_x, n_y, n_z)$ is the outward normal direction of the interface Γ, $\phi_\nu = \frac{\partial\phi}{\partial\nu}$ is the directional derivative in ν, and the notation $[f]_\Gamma = f^+ - f^-$ is the difference of the function f cross the interface Γ.

In Eqs. (1) and (2), ϵ is a piecewise function for the dielectric constants in Ω^- and Ω^+. Here κ is the inverse Debye screening length, which measures ionic strength and its modified version $\bar{\kappa}$ is given as $\bar{\kappa}^2 = \epsilon^+\kappa^2$. The value of κ is nonzero in Ω^+ only. In our numerical implementation, the unit of length is Å and the potential ϕ has units $e_c/$Å. To compute the free energy in kcal/mol/e_c, after computing ϕ it must be multiplied by a factor of 332.0716. For further details regarding the definitions of variables, coefficients, and units in the PB model we refer the reader to [18,25].



(a) (b)

Fig. 1. (a) The PB model with molecular domain Ω^- and solvent domain Ω^+ separated by the molecular surface Γ, (b) The triangulation surface of protein 6yi3, whose 2-d illustration is used in (a) as the molecular surface Γ.

The source term ρ in Eq. (1) is the summation of the charge distribution in Ω^- using delta functions for N_c partial charges located at $\mathbf{r}_i$, i.e., $\rho(\mathbf{r}) = 4\pi C \sum_{i=1}^{N_c} q_i \delta(\mathbf{r}-\mathbf{r}_i)$, where C is a constant to balance the units. Due to the singular partial charges in the source $\rho(\mathbf{r})$, the electrostatic potential $\phi(\mathbf{r})$ goes to infinity as $\mathbf{r} \to \mathbf{r}_i$.

From the description of the PB model above, it is clear that numerical solutions to the PB model face many challenges, including the complex geometry as illustrated in Fig. 1(b), interface conditions as in Eq. (2), the singular charges in ρ, the infinite computational domain in $\mathbb{R}^3$, etc. These numerical difficulties are challenging for grid-based methods, but can be conveniently addressed with the boundary integral method as described next.

2.2. Boundary integral PB solvers

In this subsection we describe the boundary integral method for computing the electrostatic surface potential and solvation free energy [17,20]. We present the boundary integral form of the PB implicit solvent model, the discretization of the boundary integral equations, the treecode algorithm for accelerating the matrix-vector product, and the preconditioning scheme to alleviate the rising condition number when the triangulation quality deteriorates due to complex geometry. Note these algorithms were described in previous work [8,20,30] and this paper emphasizes the parallelization of them using multicore CPUs and GPUs. Following the tradition of the boundary integral method, we use $\mathbf{x}$ and $\mathbf{y}$ to represent the spatial position as opposed to the previously used $\mathbf{r}$. We also denote $\Omega_1 = \Omega^-$, $\Omega_2 = \Omega^+$, $\epsilon_1 = \epsilon^-$, and $\epsilon_2 = \epsilon^+$.

2.2.1. Boundary integral form of PB model

This section summarizes the well-conditioned boundary integral form of the PB implicit solvent model we employ [20,28]. Applying Green's second identity and properties of fundamental solutions to Eq. (1) yields the electrostatic potential in each domain,

$$\phi(\mathbf{x}) = \int_\Gamma \left[G_0(\mathbf{x},\mathbf{y}) \frac{\partial \phi(\mathbf{y})}{\partial \nu} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y} + \sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad \mathbf{x} \in \Omega_1, \tag{3a}$$

$$\phi(\mathbf{x}) = \int_\Gamma \left[-G_\kappa(\mathbf{x},\mathbf{y}) \frac{\partial \phi(\mathbf{y})}{\partial \nu} + \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y}, \quad \mathbf{x} \in \Omega_2, \tag{3b}$$

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

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

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

$$\frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}) = \int_\Gamma \left[K_1(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y} + S_1(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{5a}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial \phi_1(\mathbf{x})}{\partial \nu} = \int_\Gamma \left[K_3(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_4(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y} + S_2(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{5b}$$

where $\varepsilon = \varepsilon_2/\varepsilon_1$. As given in Eqs. (6a)-(6b) 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 [20,28],

$$\begin{aligned} K_1(\mathbf{x},\mathbf{y}) &= G_0(\mathbf{x},\mathbf{y}) - G_\kappa(\mathbf{x},\mathbf{y}), \\ K_2(\mathbf{x},\mathbf{y}) &= \varepsilon \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}}, \end{aligned} \tag{6a}$$

$$\begin{aligned} K_3(\mathbf{x},\mathbf{y}) &= \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}} - \frac{1}{\varepsilon}\frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}}, \\ K_4(\mathbf{x},\mathbf{y}) &= \frac{\partial^2 G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x} \partial \nu_\mathbf{y}} - \frac{\partial^2 G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x} \partial \nu_\mathbf{y}}, \end{aligned} \tag{6b}$$

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

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{x}} = -\nu(\mathbf{x}) \cdot \nabla_\mathbf{x} G(\mathbf{x},\mathbf{y}) = -\sum_{m=1}^{3} \nu_m(\mathbf{x}) \partial_{x_m} G(\mathbf{x},\mathbf{y}), \tag{7}$$

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

$$\frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} = \nu(\mathbf{y}) \cdot \nabla_\mathbf{y} G(\mathbf{x},\mathbf{y}) = \sum_{n=1}^{3} \nu_n(\mathbf{y}) \partial_{y_n} G(\mathbf{x},\mathbf{y}), \tag{8}$$

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

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

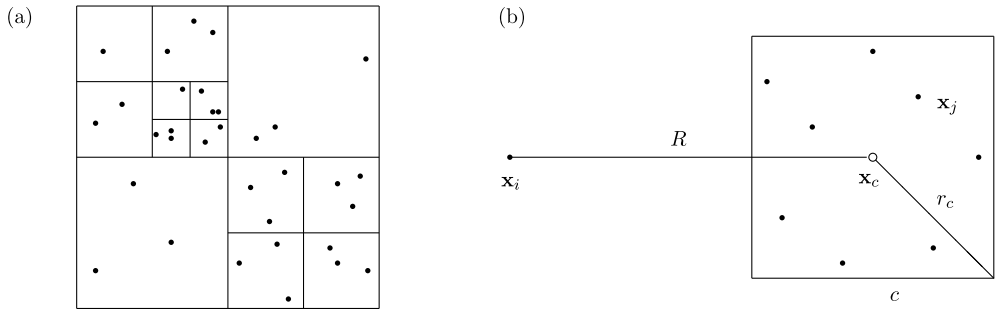


Fig. 2. Details of treecode. (a) tree structure of particle clusters. (b) particle-cluster interaction between particle $\mathbf{x}_i$ and cluster $c = \{\mathbf{x}_j\}$. $\mathbf{x}_c$: cluster center; R: particle-cluster distance; and r_c: cluster radius.

Once the potential and its normal derivative are solved from Eqs. (5a)-(5b), the potential at any point inside the molecule can be computed via Eq. (3a), or a numerically more accurate formulation may be used from [28]:

$$\phi_1(\mathbf{x}) = \int_\Gamma \left[K_1(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y} + S_1(\mathbf{x}), \quad \mathbf{x} \in \Omega_1. \tag{11}$$

With the potential and its normal derivative on Γ, the electrostatic free energy can be obtained by

$$\begin{aligned} E^{\text{free}} &= \frac{1}{2} \sum_{k=1}^{N_c} q_k \phi_1(\mathbf{y}_k) \\ &= \frac{1}{2} \sum_{k=1}^{N_c} q_k \left(\int_\Gamma \left[K_1(\mathbf{y}_k,\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{y}_k,\mathbf{y}) \phi_1(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y} + S_1(\mathbf{y}_k) \right). \end{aligned} \tag{12}$$

The electrostatic solvation free energy can also be obtained by

$$\begin{aligned} E^{\text{sol}} &= \frac{1}{2} \sum_{k=1}^{N_c} q_k \phi^{\text{reac}}(\mathbf{y}_k) \\ &= \frac{1}{2} \sum_{k=1}^{N_c} q_k \int_\Gamma \left[K_1(\mathbf{y}_k,\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu} + K_2(\mathbf{y}_k,\mathbf{y}) \phi_1(\mathbf{y}) \right] \mathrm{d}S_\mathbf{y}, \end{aligned} \tag{13}$$

where $\phi^{\text{reac}}(\mathbf{x}) = \phi(\mathbf{x}) - S_1(\mathbf{x})$ is the reaction field potential [20,28]. In our numerical results in Section 3, we focus on solution of the PB equation and calculation of the electrostatic solvation free energy.

2.2.2. Discretization of boundary integral equations

The integrals in Eqs. (5a)-(5b) can be discretized by centroid collocation, which is popular due to its simplicity [20]. Alternatively, it can be discretized using more complicated approaches such as node collocation [33,44], curved triangles [17], or Galerkin's method [11], with each resulting in a trade-off between accuracy and efficiency. Here we employ the centroid collocation approach.

Letting $\mathbf{x}_i, i = 1, \ldots, N$ denote the triangle centroids of the N triangular elements, the discretized Eqs. (5a)-(5b) have the following form for $i = 1, \ldots, N$,

$$\begin{aligned} \frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}_i) = \sum_{\substack{j=1 \\ j\neq i}}^{N} \left[K_1(\mathbf{x}_i,\mathbf{x}_j) \frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu} + K_2(\mathbf{x}_i,\mathbf{x}_j) \phi_1(\mathbf{x}_j) \right] \Delta s_j \\ + S_1(\mathbf{x}_i), \end{aligned} \tag{14a}$$

$$\begin{aligned} \frac{1}{2}\left(1+\frac{1}{\varepsilon}\right) \frac{\partial \phi_1(\mathbf{x}_i)}{\partial \nu} = \sum_{\substack{j=1 \\ j\neq i}}^{N} \left[K_3(\mathbf{x}_i,\mathbf{x}_j) \frac{\partial \phi_1(\mathbf{x}_j)}{\partial \nu} + K_4(\mathbf{x}_i,\mathbf{x}_j) \phi_1(\mathbf{x}_j) \right] \Delta s_j \\ + S_2(\mathbf{x}_i), \end{aligned} \tag{14b}$$

where Δs_j is the area of the jth boundary element, and the term $j = i$ is omitted from the summation to avoid the kernel singularity. Equations (14a)-(14b) represent 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 $S_1(\mathbf{x}_i)$ and $S_2(\mathbf{x}_i)$. We solve this system using the generalized minimal residual (GMRES) iterative method, which requires a matrix-vector product in each step [36]. Since the matrix is dense, computing the product by direct summation requires $O(n^2)$ operations, which is prohibitively expensive when n is large. These difficulties can be overcome by fast algorithms for n-body computations, such as treecode [20,46] and Fast Multipole Methods [11,32]. In the next section, we describe how the treecode algorithm is used to accelerate the matrix-vector product calculation.

2.2.3. Treecode

We summarize the treecode algorithm and refer to previous work for more details [2,15,30,31]. The matrix-vector product Ax for Eqs. (14a)-(14b) has the form of n-body potentials,

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

where G is a kernel, $\mathbf{x}_i, \mathbf{x}_j$ are centroids (also called particle locations in this context), and q_j is a charge associated with $\mathbf{x}_j$. To this end, the q_j in Eq. (15) 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. (14a)-(14b) and G is one of the kernels 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 [2]. 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, e.g. $N_0 = 3$ in Fig. 2(a)). Then V_i is evaluated as a sum of particle-particle interactions and particle-cluster interactions (depicted in Fig. 2(b)),

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

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 $\mathbf{x}_i$ [30]. Cartesian multi-index notation is used with $\mathbf{k} = (k_1, k_2, k_3), k_i \in \mathbb{N}$ and $\|\mathbf{k}\| = k_1 + k_2 + k_3$. 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 [2].

The accuracy and efficiency of the treecode is controlled by the combination of parameters including the order p, MAC parameter θ, and maximum particles per leaf N_0. Using the treecode, the operation count

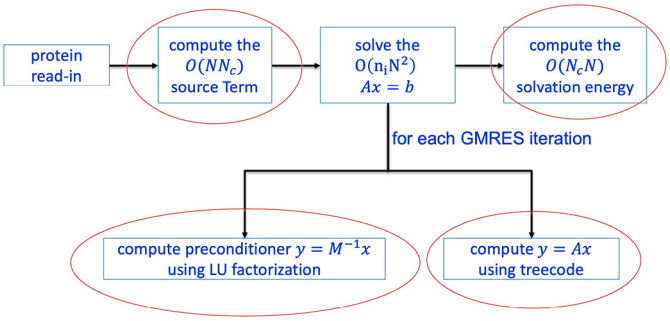


Fig. 3. Pipeline for parallelized TABI solver. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

for the matrix-vector product is $O(N \log N)$, where N is the number of particles $\mathbf{x}_i$, and the factor $\log N$ is the number of levels in the tree.

2.2.4. Preconditioning

In order to precondition Krylov subspace methods in solving $Ax = b$, we design a scheme using left-preconditioning. Given a preconditioning matrix M, we consider the modified linear system $M^{-1}Ax = M^{-1}b$. The solve then proceeds in two steps: (a) set $c = M^{-1}b$, and (b) solve $(M^{-1}A)x = c$ using GMRES. We therefore seek 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 and requires fewer GMRES iterations;
(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, thus a trade-off must be made.

We design our preconditioner based on the observation that in the 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 number of interactions, the long interactions are calculated by multipole expansions. This offers the idea that for a preconditioner of A, we may construct M to contain only short range interactions and to ignore long range interactions. To this end, we select short range interactions between elements on the same leaf only. This choice of M has great advantages in efficiency and accuracy for solving $My = z$. As seen with details in [8], by using a permutation operation, the M matrix is a block diagonal matrix with n blocks such that $M = \text{diag}\{M_1, M_2, \cdots, M_n\}$ thus $My = z$ can be solved using direct method e.g., LU factorization by solving each individual $M_i y_i = z_i$ for $i = 1, \cdots, n$. Here each M_i is a square nonsingular matrix, which represents the interaction between particles/elements on the ith leaf of the tree. It is worthy to note that the efficiency is not affected even when M_i has a large condition number since a direct solver is used for solving $My = z$. Meanwhile, the computational cost for solving $My = z$ is $O(Nn_0^2)$ with n_0 the treecode parameter, maximum number of particles per leaf, as detailed in [8].

2.3. MPI-based parallelization of the TABI solver

As illustrated by the red circled items in Fig. 3, our parallelization of the TABI solver focuses on the four stages of the pipeline: the $O(NN_c)$ source term, $O(n_i N \log N)$ matrix-vector product using treecode, the $O(n_i Nn_0^2)$ preconditioner, and the $O(N_c N)$ solvation energy. Here, N is the number of surface triangles, n_0 is the maximum number of particles per leaf, N_c is number of partial charges, and n_i is the number of GMRES iterations. Among these stages, the most time consuming and challenging component is the matrix-vector product using treecode. To this end, we investigate two possible strategies for computing n-body problems as in [10], which are also briefly described below. In this work, we migrate these strategies to solving the boundary integral PB model. The numerical results in Section 3 show high parallel efficiency from the optimized approach.

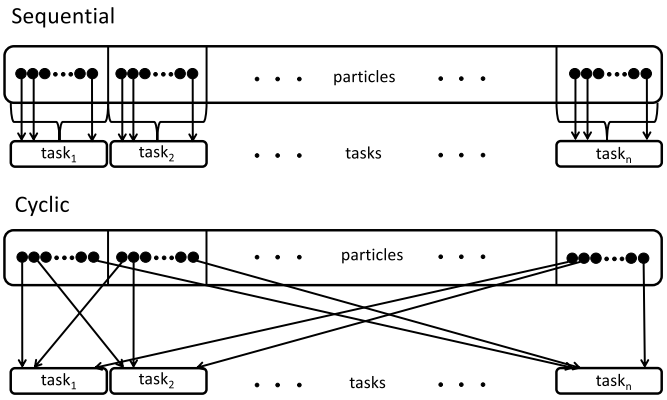


Fig. 4. Methods for assigning target particles to tasks: sequential order (top) vs cyclic order (bottom).

The initial and intuitive method to assign target particles to tasks is to use *sequential ordering*, in which the 1st task handles the first N/n_p particles in a consecutive segment, the 2nd task handles the next N/n_p particles, etc. The illustration of this job assignment is shown in the top of Fig. 4. However, when examining the resulting CPU time on each task, we noticed starkly different times on each task, indicating a severe load imbalance. This may be understood by the fact that for particles at different locations, the types of interactions with the other particles through the tree can vary. For example, a particle with only a few close neighbors uses more particle-cluster interactions than particle-particle interactions, thus requiring less CPU time than a particle with many close neighbors. We also notice that for particles that are nearby one another, their interactions with other particles, either by particle-particle interaction or particle-cluster interaction, are quite similar, so some consecutive segments ended up computing many more particle-particle interactions than others that were instead dominated by particle-cluster interactions. Based on these observations, we designed a *cyclic ordering* scheme to improve load balancing, as illustrated on the bottom of Fig. 4. In this scheme, particles nearby one another are uniformly distributed to different tasks. For example, for a group of particles close to each other, the first particle is handled by the first task, the second particle is handled by the second task, etc. The cycle repeats starting from the $(n_p + 1)$-th particle. The numerical results that follow demonstrate the significantly improved load balance from this simple scheme.

The pseudocode for our MPI-based parallel TABI solver using replicated data algorithm is given in Table 1. The identical trees are built on each task as in line 6. The four-stage MPI-based parallelization of the source term, matrix-vector product with treecode, preconditioning, and solvation energy occur in lines 7, 10, 12, and 17, respectively, followed by MPI communications.

2.4. The GPU-accelerated DSBI solver

The pseudocode for the DSBI-PB solver using GPUs is given in Table 2. In this pseudocode, we divide all the operations into those on host performed by the CPUs and those on device performed by the GPUs. The three compute-intensive stages are computation of the source term, matrix-vector product, and solvation energy; each are computed on GPUs as shown in lines 6, 11, and 20, followed by a copy of the data from device to host. The host CPU takes care of all complicated and non-concurrent work. We note that lines 13 and 14 are still under investigation due to the considerations of parallel efficiency, and we disable these two lines in our current numerical implementation.

Table 1
Pseudocode for MPI-based parallel TABI solver using replicated data algorithm.

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

Table 2
Pseudocode for DSBI-PB solver using GPU.

```
On host (CPU)
    read biomolecule data (charge and structure)
    call MSMS to generate triangulation
    copy biomolecule data and triangulation to device
On device (GPU)
    each thread concurrently computes and stores source terms for assigned triangles
    copy source terms on device to host
On host
    set initial guess x_0 for GMRES iteration and copy it to device
On device
    each thread concurrently computes assigned segment of matrix-vector product y = Ax
    copy the computed matrix-vector y to host memory
*   each thread concurrently solves its assigned portion of Mx = y
*   copy the solution x to host memory
On host
    test for GMRES convergence
        if no, generate new x and copy it to device, go to step 10 for the next iteration
        if yes, generate and copy the final solution to device and go to step 19
On device
    compute assigned segment of electrostatic solvation free energy
    copy computed electrostatic solvation free energy contributions to host
On host
    add segments of electrostatic solvation free energy and output result
```

* currently disabled

One challenge is that the variance in sizes of the block matrices M_i for $i = 1, \ldots, n$ that compose the preconditioner M lead to significant load imbalance on the GPU. The other is that the LU factorization used for solving the $M_i y_i = z_i$ is very inefficient on GPUs for our current implementation. However, disabling the preconditioner within the GPU based parallelization could significantly increase computing time when A is ill-conditioned.

2.5. COVID-19 proteins

Coronaviruses are a persistent threat to global health. Viruses such as SARS in 2003, MERS in 2013, and the new SARS-CoV-2 in 2019 emerge from animal populations and then infect humans. Coronaviruses contain a large genome which directs the synthesis of several dozen viral proteins. Structures of these proteins are used to better understand

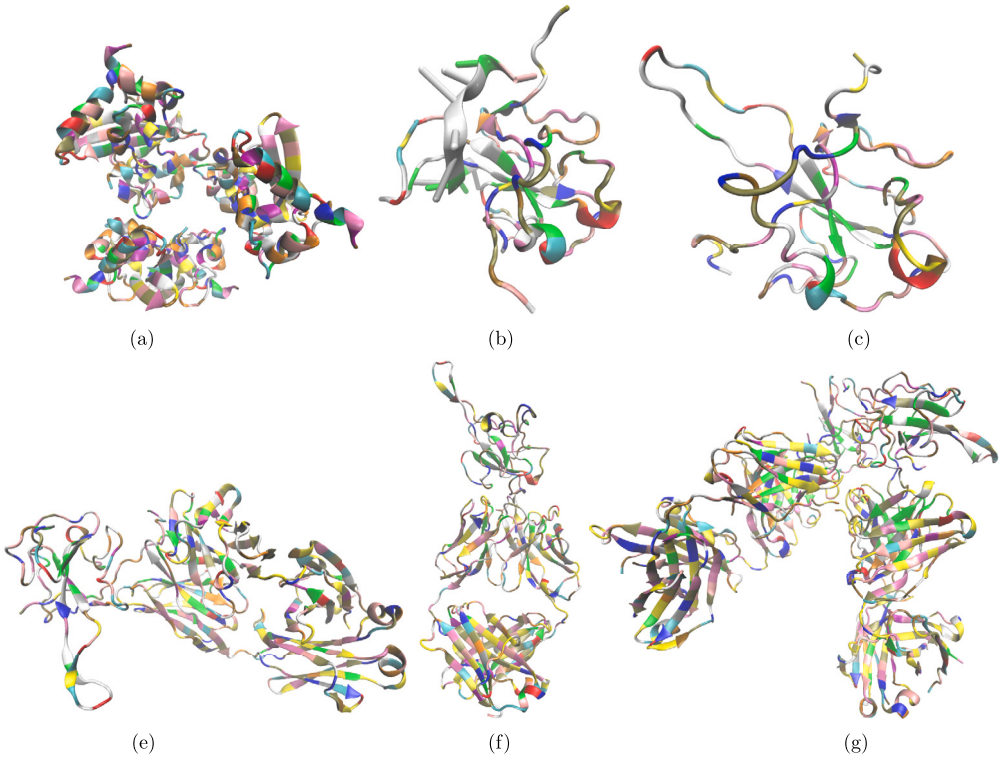


Fig. 5. COVID-19 related proteins used in our numerical simulations. (a) **6wji**: The C-terminal Dimerization Domain of Nucleocapsid Phosphoprotein from SARS-CoV-2; (b) **7act**: SARS-CoV-2 Nucleocapsid Phosphoprotein N-Terminal Domain in Complex with 10mer ssRNA; (c) **6yi3**: The N-terminal RNA-Binding Domain of the SARS-CoV-2 Nucleocapsid Phosphoprotein; (d) **7cr5**: Human Monoclonal Antibody with SARS-CoV-2 Nucleocapsid Protein NTD; (e)**7n3c**: Human Fab S24-202 in the Complex with the N-Terminal Domain of Nucleocapsid Protein from SARS-CoV-2; (f) **7sts**: Human Fab S24-1379 in the Complex with the N-terminal Domain of Nucleocapsid Protein from SARS-CoV-2.

the diseases, and to develop new drugs and vaccines to fight coronaviruses. In this work, we focus on proteins involved in the spreading and prevention of the COVID-19 virus. The virus genome in the form of an mRNA encodes proteins including replication/transcription complexes that make more RNA, structural proteins that construct new virions, and proteases (e.g. 61u7) that cut polyproteins into all of these functional pieces. The virus docks to target cells by binding the spike protein (e.g. proteins 6crz, 6vxx, 6vsp, 6vsb) on the viral surface to its receptor, angiotensin-converting enzyme 2 (ACE2, e.g. protein 6m17) on the target cell membrane. In addition, to test the infection of COVID-19 virus, we often identify its nucleocapsid proteins (e.g. proteins 7act, 6yi3) by using antibodies (e.g. proteins 7cr5, 7n3c, 7sts) that particularly bind to these nucleocapsids [1].

In this work, we select a few COVID-19 related proteins and use our parallel PB solvers to calculate their electrostatic properties such as global solvation energy or local surface potential. These protein electrostatics can assist researchers in understanding a protein's overall structure and function, their binding affinity to certain ligands, as well as their folding and enzyme catalysis characteristics.

In Fig. 5, we provide the cartoon structure of six COVID-19 related proteins. Protein 6wji [41] in (a) is a dimerization domain, which is used to bring two nucleocapsids together. The connection of nucleocapsid dimers into bigger groups makes the viral structure that encases the RNA in the limited area within virus particles. The SARS-CoV-2 nucleocapsid contains separate proteins which all perform different functions. A portion of the structure folds into an RNA-binding domain (protein 7act) [13] as shown in (b), featuring a groove that securely holds a brief segment of the viral genomic RNA. In contrast, the protein alone without the RNA-bound structure (protein 6yi3) is shown in (c) [13]. In COVID-19 prevention, home test kits for detecting SARS-CoV-2 infection rely on antibody proteins that specifically recognize nucleocapsids within a complex set of biomolecules in nasal samples. The antibodies they recognize differ based on the test-kit brand, which recognize different portions of the nucleocapsid, and we list a few here with protein 7cr5[29] in (d), protein 7n3c in (e), and protein 7sts in (f). For these listed proteins, the PB model is solved using our parallel boundary integral PB solvers and numerical results are presented in the following section.

3. Numerical results

Our numerical results are generated on supercomputers sponsored by Southern Methodist University's Center for Research Computing (CRC). The MPI-based results are generated on M3 (https://www.smu.edu/oit/services/m3) and GPU-based results are generated on SuperPOD (https://www.smu.edu/oit/services/superpod).

3.1. Parallel efficiency of MPI-based computing

We first check the parallel efficiency of our MPI-based algorithm with both sequential and cyclic schemes by computing the solvation energy on protein 7n3c at MSMS density of 12, which generates 529,911 boundary elements. We use up to 256 MPI tasks and Table 3 shows the results. Column 1 shows the increasing numbers of MPI tasks. Column 2 reports the total CPU time when the direct sum (DS)BI scheme is used to compute the electrostatic solvation free energy. Due to its $O(N^2)$ computational cost, the CPU time for the DSBI solver is overly long, even when 256 tasks are used. Columns 4 and 5 display the total CPU time and parallel efficiency for the TABI solver using the sequential and cyclic schemes, both of which are much faster compared to DSBI. Columns 8 and 9 focus more closely on the time required for a single matrix-vector product Ax, $\bar{t}_{Ax}$, which we take as the average of the iteration's maximum CPU time among all tasks,

Table 3
CPU time and parallel efficiency (P.E.) for parallelized direct sum, sequentially parallelized treecode (seq.) and cyclically parallelized treecode (cyc.) for computing electrostatic solvation energy (-6020.52 kcal/mol from TABI solver and -6013.68 kcal/mol from DSBI solver) for protein 7n3c with 529,955 boundary elements. The treecode parameters are $\theta = 0.8$, $N_0 = 100$, and $p = 3$; The number of tasks n_p ranges over $1, \ldots, 256$. The time for one Ax ($\bar{t}_{Ax}$) is the average iteration's maximum CPU time over all tasks.

n_p	DSBI Solver		TABI solver							
	Total Time		Total Time				Time for one Ax ($\bar{t}_{Ax}$,)			
	CPU (s)	P.E. (%)	CPU (s)		P.E. (%)		CPU (s)		P.E. (%)	
			seq.	cyc.	seq.	cyc.	seq.	cyc.	seq.	cyc.
1	106063.17	100.00	1874.88	1873.60	100.00	100.00	89.75	89.60	100.00	100.00
2	53132.86	99.81	971.25	967.12	96.52	96.87	45.49	45.22	98.63	99.07
4	26549.87	99.87	561.25	502.60	83.51	93.20	25.69	22.57	87.34	99.26
8	13291.47	99.75	321.42	285.25	72.91	82.10	13.94	12.02	80.46	93.22
16	6710.06	98.79	171.41	158.04	68.36	74.09	6.43	5.77	87.30	97.08
32	3928.71	84.37	128.13	114.73	45.73	51.03	3.99	3.26	70.22	85.84
64	2022.84	81.93	99.75	90.81	29.37	32.24	2.14	1.66	65.55	84.24
128	1042.49	79.48	79.82	76.83	18.35	19.05	1.08	0.85	64.65	82.77
256	554.50	74.72	71.70	71.16	10.21	10.28	0.56	0.45	62.27	78.61

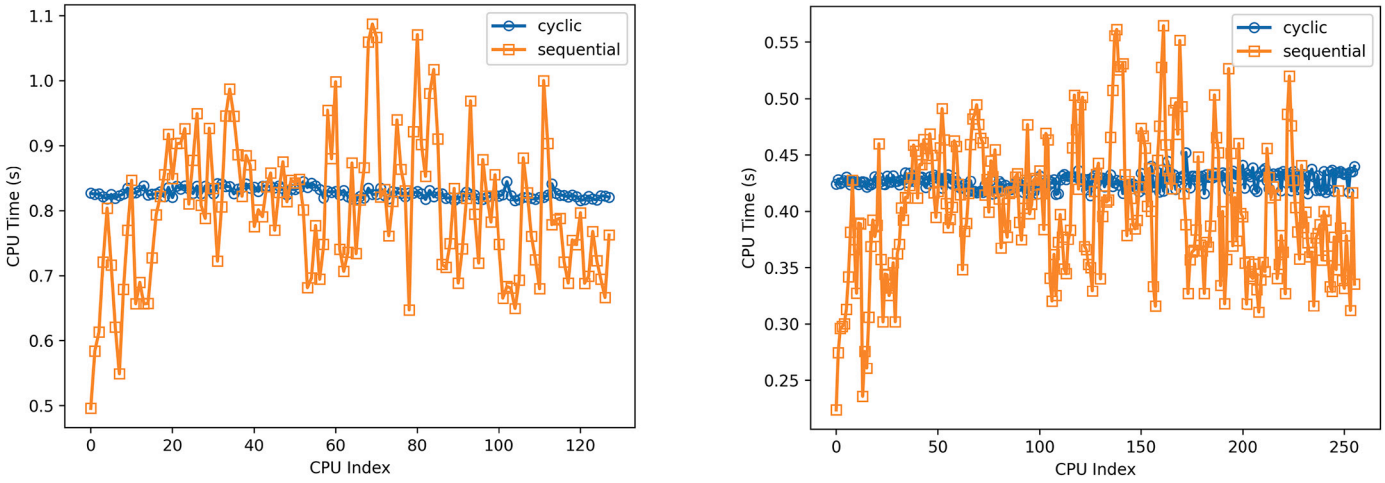


Fig. 6. MPI-based parallelization with sequential and cyclic schemes: left: 128 tasks, right: 256 tasks. The CPU time reported is $\bar{t}_{Ax}$, the averages GMRES iteration's maximum CPU times among all tasks.

$$\bar{t}_{Ax} = \frac{1}{n_i} \sum_{k=1}^{n_i} \max_j t_{Ax}^{j,k} \tag{17}$$

where $t_{Ax}^{j,k}$ is the CPU time to compute Ax from the jth task in the kth GMRES iteration.

Parallel efficiencies are displayed in Columns 3, 6, 7, 10 and 11. The parallelization of the DSBI solver shows high efficiency as seen in column 3. This is due to the simplicity of the algorithm. Other than the four stages identified in Fig. 3, there is very little serial computation or communication required. However, the parallel efficiency of the TABI solver is not as good as the DSBI solver, as shown in Columns 6 and 7. This is primarily due to the use of treecode, which has some serial time for building the tree and computing the moments. The serial time is relatively short when n_p is small but becomes increasingly significant as n_p grows and the time spent within parallelized stages decreases. If we focus specifically on the parallelization of the treecode in computing Ax, Columns 10 and 11 show a high degree of parallel efficiency. We can also observe that the cyclic scheme significantly improves the parallel efficiency in comparison with the sequential scheme. However, due to the very small fraction of runtime spent in computing matrix-vector products as n_p increases, the overall parallel efficiencies from Columns 6 and 7 do not show a significant difference between the sequential and cyclic schemes. To more carefully examine the performance differences between the sequential and cyclic schemes, in Fig. 6 we plot $\bar{t}_{Ax}$ from Eq. (17) when 128 and 256 MPI tasks are used. It is evident that the cyclic scheme has reduced variance compared with the sequential scheme owing to its advantage in load balance.

3.2. MPI-based TABI solver vs GPU-accelerated DSBI solver

Next, we compute the solvation energy for the six COVID-19 proteins introduced previously using MSMS with density equal to 12 to provide sufficient detail of the molecular surface. We use both the MPI-based TABI solver and the GPU-accelerated DSBI solver. For a reasonable computing power comparison, we use 64 CPU cores for the MPI-related computing and 1 GPU card for the GPU-related computing. Table 4 shows the simulation results. Column 1 is the PDB ID for proteins in ascending sequence of their size followed by the number of atoms in column 2, number of boundary elements in column 3, and the areas of the solvent excluded surface in column 4. Columns 5 and 6 are the number of GMRES iterations, from which we can see that the TABI solver has much improved condition number in comparison with the DSBI solver thanks to the TABI preconditioner from Section 2.2.4. The solvation energies are reported in columns 7 and 8, which are suffi-

Table 4
Computing electrostatic solvation energies in (kcal/mol) for the involved proteins: ionic strength = 0.15M; $\epsilon_1 = 1$, $\epsilon_2 = 80$; MSMS [38] density = 12; N_c is the number of atoms/charges, N is the number of boundary elements, n_i is the number of GMRES iterations, S_{ses} is the solvent excluded surface area in the units of Å^2, and E_{sol} is the electrostatic solvation energy; For the MPI-based TABI solver, the treecode parameters are $\theta = 0.8$, $N_0 = 100$, and $p = 3$.

PDB	N_c	N	S_{SES}	n_i^{MPI}	n_i^{GPU}	E_{sol}^{MPI}	E_{sol}^{GPU}	t^{MPI} (s)	t^{GPU} (s)
6yi3	2083	169,955	7516.44	10	10	-1941.81	-1945.18	14.76	8.96
7act	2352	188,027	8286.70	14	14	-1893.88	-1934.49	21.35	17.39
7cr5	8133	513,139	22524.23	16	100+	-5713.52	-5786.69	89.52	695.17
7n3c	8459	529,955	23244.85	19	17	-6020.52	-6013.68	99.13	132.20
6wji	10182	641,155	28116.88	13	14	-14009.55	-14016.02	112.82	152.71
7sts	15797	993,461	43457.63	26	100+	-11622.63	-11583.26	422.67	2544.70

Table 5
Computing electrostatic solvation energies in (kcal/mol) for the protein 6yi3 at different MSMS densities: ionic strength = 0.15M; $\epsilon_1 = 1$, $\epsilon_2 = 80$; d is the MSMS density, N is the number of boundary elements, n_i is the number of GMRES iterations, E_{sol} is the electrostatic solvation energy. Results are generated using KOKKOS and MPI on ManeFrame III; MPI results are from using 64 tasks; GPU results are from using one A100 GPU; For the MPI-based TABI solver, the treecode parameters are $\theta = 0.8$, $N_0 = 100$, and $p = 3$.

d	N	E_{sol}^{MPI}	E_{sol}^{GPU}	n_i^{CPU}	n_i^{GPU}	t_{CPU} (s)	t_{MPI} (s)	t_{GPU} (s)
2	28,767	-2057.61	-2056.26	10	10	29.34	2.76	0.83
4	56,127	-1999.01	-1997.03	10	10	66.40	4.46	1.52
6	84,903	-1968.00	-1966.87	10	16	108.52	7.05	4.98
8	110,307	-1954.62	-1952.23	10	10	145.13	9.35	4.32
12	169,955	-1945.18	-1941.81	10	10	240.54	14.76	8.91
16	229,901	-1940.96	-1936.87	10	11	340.82	19.86	17.66
18	257,236	-1938.27	-1933.67	10	11	385.96	**21.69**	**23.73**
20	287,202	-1937.18	-1931.77	10	12	438.86	24.54	28.73
24	343,806	-1933.63	-1928.65	10	11	534.31	35.13	38.62
28	407,196	-1933.04	-1927.56	10	12	653.06	41.84	55.18
32.5	471,307	-1931.76	-1926.04	10	12	760.12	51.23	77.83
64	946,335	-1928.51	-1921.81	10	13	1701.06	145.65	311.07

ciently close. The differences are caused by the treecode approximation, the preconditioning scheme, and the error tolerance achieved when the iteration is stopped. Note for calculating protein electrostatic solvation energy, we don't have an exact value to compare. If the DSBI solver converges before reaching the maximum number of allowed GMRES iterations, its result should be more accurate than that from TABI solver since treecode could add extra approximations. For example, the E_{sol}^{GPU} results from proteins 6yi3, 7act, 7n3c, 6wji should be more accurate than the E_{sol}^{MPI} results for these proteins. However, if the GMRES allowed maximum number of iteration has been reached for examples for proteins 7cr5 and 7sts, on which the DSBI solver stopped when 100 iterations are reached while the accuracy did not meet the 10^{-4} threshold, we can not say for sure whether E_{sol}^{GPU} result or E_{sol}^{MPI} result is more accurate. The computation times are shown in columns 9 and 10, which demonstrates that the computing power between 64 CPUs and 1 GPU are comparable. However, the algorithms (with preconditioning vs without preconditioning, direct sum vs treecode) can make a substantial difference for ill-conditioned or larger systems. For example, for protein 7sts, with nearly one million boundary elements, the MPI-based TABI solver is significantly faster than the GPU-based DSBI because of the ill-conditioned system, the large size of the problem, and the use of treecode or not.

We then further investigate under what conditions we should choose between using the GPU-accelerated DSBI solver or MPI-based TABI solver. The following example, whose result is shown in Table 5, gives some important guidance. In this example, we compute the solvation energy for protein 6yi3 for increasing values of the MSMS density (d), giving rise to increased problem sizes, as shown in columns 1 and 2. Columns 3 and 4 show the similar solvation energy computed with these two approaches. Columns 5 and 6 report the number of GMRES iterations. From these close results, we see that the discretized system for this protein is well conditioned thus the preconditioning scheme has limited effect. We solve the problem using 1 CPU core and report the time in column 7 for reference. Then we report the time for solving the problem using 64 MPI tasks and one A100 GPU card in columns 8 and 9. The result indicates that for a protein whose discretized system is well-conditioned, when the number of boundary elements is less than 250,000, we should use the GPU-accelerated DSBI solver, since the smaller the system the better the GPU-accelerated DSBI solver compares against the MPI-based TABI solver. Note: this "250,000" break-even value is suggested for the current hardware conditions (e.g. the 64 AMD EPYC 7763 CPU cores vs one A100 Nvidia GPU card) and algorithm configuration (e.g. treecode parameters are $\theta = 0.8$, $N_0 = 100$, and $p = 3$). Changes in computational hardware or treecode parameters will affect this precise break-even value, although the overall trend will remain unchanged. In addition, if the conditioning of A shows a pressing need for preconditioning, the threshold number will be smaller for the GPU-accelerated DSBI solver. The rapid GPU performance at least gives us the hope to perform molecular dynamics or Monte Carlo simulation for small and middle-sized proteins using GPUs. For example, if 50,000 boundary elements can reasonably describe the given protein, a single PB equation solution only takes about one second using one GPU card, in comparison with 4 seconds on a 64-core cluster.

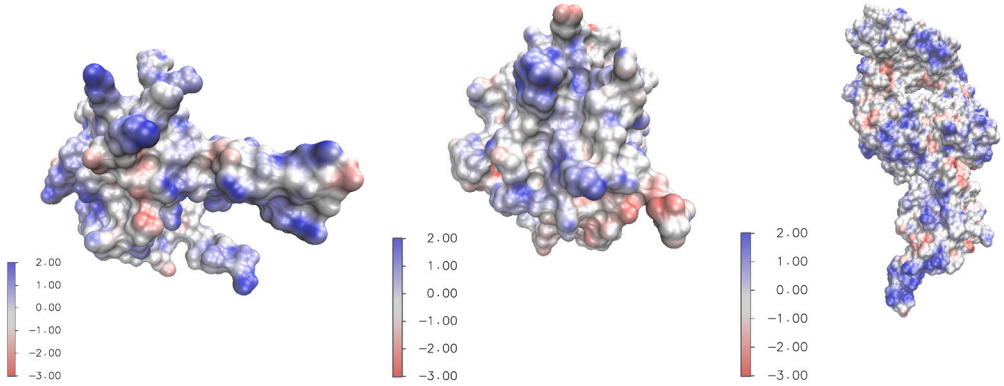


Fig. 7. Color coded electrostatic surface potential in kcal/mol/e_c on the molecular surface of proteins 6yi3 (left), 7act (middle), and 7n3c (right); plot is drawn with VMD [27].

We note that the solution to the boundary integral PB equation gives both the electrostatic potential and its normal derivative on the molecular surface. We can plot the potential on the surface elements. The color-coded potential can provide guidance on the docking site for the ligand, or offer other insights pertaining to protein-protein interactions. Some examples of this kind of visualization are shown in Fig. 7.

4. Software dissemination

The source code for all solvers and examples used in this project are available on GitHub. The MPI code can be found on https://github.com/elyssasliheet/tabi_mpi_code maintained by SMU graduate student Elyssa Sliheet. The GPU code can be found on https://github.com/yangxinsharon/bimpb-parallelization maintained by SMU graduate student Xin Yang.

5. Concluding remarks

In this project, we investigate the practical application of the PB model on selected proteins which play significant roles in the spread, treatment, and prevention of COVID-19 virus diseases. To this end, we solved the boundary integral form of the PB equation on the molecular surfaces of these proteins. These calculations produce both the electrostatic solvation free energy as a global measurement and the electrostatic surface potential for local details of the selected proteins. We investigated the parallel performance of two competing solvers for the boundary integral PB equations on these selected proteins. By considering the advantages of current algorithms and computer hardware, we focused on the parallelization of the TABI solver using MPI on CPUs and the DSBI solver using KOKKOS on GPUs. Our numerical simulations show that the DSBI solver on one A-100 GPU is faster than the TABI solver with MPI on 64 CPUs when the number of elements is smaller than 250,000. When both GPU and MPI are available and the triangulation quality is good enough so that the TABI preconditioner is not needed for GMRES convergence, we recommend that the GPU-accelerated DSBI PS solver be used when the number of boundary elements is below 250,000. Otherwise, the MPI-based TABI should be used since the TABI and DSBI algorithms require $O(N \log N)$ and $O(N^2)$ operations, respectively. If the number of elements becomes so large such that the memory on a CPU task cannot hold an entire tree, we recommend consideration of a domain-decomposition MPI scheme [9,37,42,46]. We note that the memory usage for TABI scales linearly with problem size. When one million boundary elements are used, the memory usage is a little bit over 1 GB. Thus for popular tasks on clusters with at least 64G memory per MPI rank, we can handle problems as large as approximately 64 million boundary elements, which is sufficient for simulating middle-large proteins with up to tens of thousands atoms. For even larger biomolecules, e.g. the viral capsids of Zika or H1N1 virus with up to tens of millions atoms [44], domain decomposition approach can be considered [46].

CRediT authorship contribution statement

Xin Yang: Visualization, Software, Data curation. **Elyssa Sliheet:** Writing – review & editing, Visualization, Software, Data curation. **Reece Iriye:** Visualization, Data curation. **Daniel Reynolds:** Writing – review & editing, Supervision, Resources. **Weihua Geng:** Writing – review & editing, Writing – original draft, Project administration, Methodology, Funding acquisition, Data curation, Conceptualization.

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 grants DMS-2110922 and DMS-2110869. Elyssa Sliheet reports financial support was provided by National Science Foundation grants RTG-1840260 and DMS-2110869. Xin Yang reports financial support was provided by National Science Foundation grants DMS-2110922. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work of XY, ES, and WG was supported in part by the National Science Foundation (NSF) grants DMS-2110922 and DMS-2110869. ES was also partially supported by the NSF RTG-1840260 grant. RI was also supported in part by SMU's Hamilton Scholar and Undergraduate Research Assistantships (URA) programs.

We thank the SMU Mathematics Department for providing the parallel computing class MATH 6370, which systematically trains graduate students on parallelization strategies, schemes, and experience. We also thank the SMU Center for Research Computing (CRC) for proving computing hardware. These resources combined to make this project possible.

References

[1] http://www.rcsb.org/pdb/home/home.do.
[2] J. Barnes, P. Hut, A hierarchical O(NlogN) force-calculation algorithm, Nature 324 (1986) 446–449, https://doi.org/10.1038/324446a0.

[3] 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.
[4] J. Bédorf, E. Gaburov, S. Portegies Zwart, A sparse octree gravitational N-body code that runs entirely on the GPU processor, J. Comput. Phys. 231 (7) (2012) 2825–2839.
[5] R.G. Belleman, J. Bédorf, S.F. Portegies Zwart, High performance direct gravitational N-body simulations on graphics processing units II: an implementation in CUDA, New Astron. 13 (2) (2008) 103–112.
[6] M. Burtscher, K. Pingali, An Efficient CUDA Implementation of the Tree-Based Barnes-Hut N-Body Algorithm, Elsevier Inc., 2011, pp. 75–92.
[7] 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 (9) (2010) 1–12, https://doi.org/10.1371/journal.pone.0012722.
[8] J. Chen, W. Geng, On preconditioning the treecode-accelerated boundary integral (TABI) Poisson-Boltzmann solver, J. Comput. Phys. 373 (2018) 750–762.
[9] J. Chen, W. Geng, D. Reynolds, Cyclically paralleled treecode for fast computing electrostatic interactions on molecular surfaces, Comput. Phys. Commun. 260 (2021) 107742.
[10] J. Chen, W. Geng, G.W. Wei, MLIMC: machine learning-based implicit-solvent Monte Carlo, Chin. J. Chem. Phys. 34 (6) (2021) 683–694, https://doi.org/10.1063/1674-0068/cjcp2109150.
[11] J. Chen, J. Tausch, W. Geng, A Cartesian FMM-accelerated Galerkin boundary integral Poisson-Boltzmann solver, J. Comput. Phys. 478 (2023) 111981.
[12] 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 beta2-adrenergic G protein–coupled receptor, Science 318 (5854) (2007) 1258–1265, https://doi.org/10.1126/science.1150577, http://science.sciencemag.org/content/318/5854/1258.
[13] D.C. Dinesh, D. Chalupska, J. Silhan, E. Koutna, R. Nencka, V. Veverka, E. Boura, Structural basis of RNA recognition by the SARS-CoV-2 nucleocapsid phosphoprotein, PLoS Pathog. 16 (12) (2020) 1–16.
[14] 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, http://www.biophysj.org/cgi/content/abstract/85/1/49.
[15] 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, https://doi.org/10.1002/1096-987X(20010130)22:2<184::AID-JCC6>3.0.CO;2-7.
[16] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, V. Pande, N-body simulation on GPUs, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 188–es.
[17] W. Geng, Parallel higher-order boundary integral electrostatics computation on molecular surfaces with curved triangulation, J. Comput. Phys. 241 (2013) 253–265, https://doi.org/10.1016/j.jcp.2013.01.029.
[18] W. Geng, A boundary integral Poisson–Boltzmann solvers package for solvated biomolecular simulations, Comput. Math. Biophys. 3 (2015) 43–58.
[19] W. Geng, F. Jacob, A GPU-accelerated direct-sum boundary integral Poisson-Boltzmann solver, Comput. Phys. Commun. 184 (6) (2013) 1490–1496, https://doi.org/10.1016/j.cpc.2013.01.017.
[20] W. Geng, R. Krasny, A treecode-accelerated boundary integral Poisson-Boltzmann solver for electrostatics of solvated biomolecules, J. Comput. Phys. 247 (2013) 62–78, https://doi.org/10.1016/j.jcp.2013.03.056.
[21] W. Geng, S. Yu, G.W. Wei, Treatment of charge singularities in implicit solvent models, J. Chem. Phys. 127 (2007) 114106.
[22] 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.
[23] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (2) (1987) 325–348, https://doi.org/10.1016/0021-9991(87)90140-9.
[24] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri, M. Taiji, A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation, Comput. Sci. Res. Dev. 24 (1) (2009) 21–31, https://doi.org/10.1007/s00450-009-0089-1.
[25] M.J. Holst, The Poisson-Boltzmann equation: Analysis and multilevel numerical solution, Ph.D. thesis, UIUC, 1994.
[26] 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, https://doi.org/10.1126/science.1222804.
[27] W. Humphrey, A. Dalke, K. Schulten, VMD – visual molecular dynamics, J. Mol. Graph. 14 (1) (1996) 33–38, https://doi.org/10.1016/0263-7855(96)00018-5.
[28] 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.
[29] S. Kang, M. Yang, S. He, Y. Wang, X. Chen, Y.Q. Chen, Z. Hong, J. Liu, G. Jiang, Q. Chen, Z. Zhou, Z. Zhou, Z. Huang, X. Huang, H. He, W. Zheng, H.X. Liao, F. Xiao, H. Shan, S. Chen, A SARS-CoV-2 antibody curbs viral nucleocapsid protein-induced complement hyperactivation, Nat. Commun. 12 (1) (2021) 2697.
[30] P. Li, H. Johnston, R. Krasny, A Cartesian treecode for screened Coulomb interactions, J. Comput. Phys. 228 (2009) 3858–3868, https://doi.org/10.1016/j.jcp.2009.02.022.
[31] 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, https://doi.org/10.1006/jcph.2001.6862.
[32] 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, https://doi.org/10.1016/j.jcp.2007.05.026.
[33] B. Lu, J.A. McCammon, Improved boundary element methods for Poisson-Boltzmann electrostatic potential and force calculations, J. Chem. Theory Comput. 3 (2007) 1134–1142, https://doi.org/10.1021/ct700001x, PMID: 26627432.
[34] 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, https://doi.org/10.1002/jcc.24757.
[35] L. Nyland, M. Harris, J. Prins, Fast N-Body Simulation with CUDA, GPU Gems, vol. 3, Addison-Wesley, Upper Saddle River, NJ, 2009.
[36] Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869, https://doi.org/10.1137/0907058.
[37] J.K. Salmon, M.S. Warren, G.S. Winckelmans, Fast parallel tree codes for gravitational and fluid dynamical n-body problems, Int. J. Supercomput. Appl. 8 (1986) 129–142.
[38] M.F. Sanner, A.J. Olson, J.C. Spehner, REDUCED SURFACE: an efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.
[39] T. Simonson, G. Archontis, M. Karplus, Free energy simulations come of age: protein-ligand recognition, Acc. Chem. Res. 35 (6) (2002) 430–437, https://doi.org/10.1021/ar010030m, PMID: 12069628.
[40] N. Unwin, Refined structure of the nicotinic acetylcholine receptor at 4Å resolution, J. Mol. Biol. 346 (4) (2005) 967–989, https://doi.org/10.1016/j.jmb.2004.12.031.
[41] J.P. Vandervaart, N.L. Inniss, T. Ling-Hu, G. Minasov, G. Wiersum, M. Rosas-Lemus, L. Shuvalova, C.J. Achenbach, J.F. Hultquist, K.J.F. Satchell, K.E.R. Bachta, Serodominant SARS-CoV-2 nucleocapsid peptides map to unstructured protein regions, Microbiol. Spectr. 11 (3) (2023) e0032423.
[42] N. Vaughn, L. Wilson, R. Krasny, A GPU-accelerated barycentric Lagrange treecode, in: 2020 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2020, pp. 701–710.
[43] 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, https://doi.org/10.1073/pnas.0600118103.
[44] 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 (37) (2022) 7104–7113, https://doi.org/10.1021/acs.jpcb.2c04604.
[45] L. Wilson, J. Hu, J. Chen, R. Krasny, W. Geng, Computing electrostatic binding energy with the TABI Poisson–Boltzmann solver, Commun. Inf. Syst. 22 (2) (2022) 247–273.
[46] L. Wilson, N. Vaughn, R. Krasny, A GPU-accelerated fast multipole method based on barycentric Lagrange interpolation and dual tree traversal, Comput. Phys. Commun. 265 (2021) 108017.
[47] B. Zhang, B. Peng, J. Huang, N.P. Pitsianis, X. Sun, B. Lu, Parallel AFMPB solver with automatic surface meshing for calculation of molecular solvation free energy, Comput. Phys. Commun. 190 (2015) 173–181, https://doi.org/10.1016/j.cpc.2014.12.022.
[48] Y.C. Zhou, B. Lu, A.A. Gorfe, Continuum electromechanical modeling of protein-membrane interactions, Phys. Rev. E 82 (2010) 041923, https://doi.org/10.1103/PhysRevE.82.041923.