THE JOURNAL OF PHYSICAL CHEMISTRY B
A JOURNAL OF THE AMERICAN CHEMICAL SOCIETY

pubs.acs.org/JPCB

Article

TABI-PB 2.0: An Improved Version of the Treecode-Accelerated Boundary Integral Poisson-Boltzmann Solver

Published as part of The Journal of Physical Chemistry virtual special issue "Biomolecular Electrostatic Phenomena".

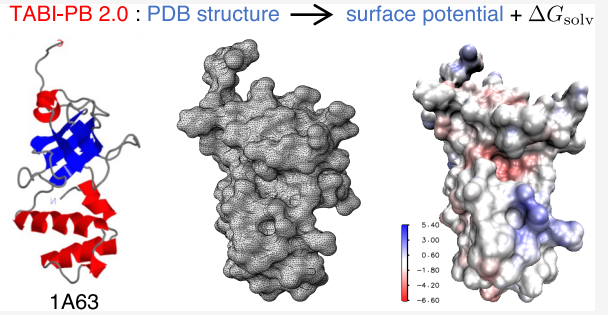
Leighton Wilson,* Weihua Geng,* and Robert Krasny*

Cite This: *J. Phys. Chem. B* 2022, 126, 7104–7113

Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: This work describes TABI-PB 2.0, an improved version of the treecode-accelerated boundary integral Poisson-Boltzmann solver. The code computes the electrostatic potential on the molecular surface of a solvated biomolecule, and further processing yields the electrostatic solvation energy. The new implementation utilizes the NanoShaper surface triangulation code, node-patch boundary integral discretization, a block preconditioner, and a fast multipole method based on barycentric Lagrange interpolation and dual tree traversal. Performance-critical portions of the code were implemented on a GPU. Numerical results for protein 1A63 and two viral capsids (Zika, H1N1) demonstrate the code's accuracy and efficiency.



INTRODUCTION

Implicit solvent models play an important role in describing electrostatic interactions between biomolecules and their solvent environment.[1−3] Of particular interest is the Poisson-Boltzmann (PB) model[4,5] comprising a domain $\Omega_1 \subset R^3$ with dielectric constant ε_1 containing the solute biomolecule and a domain $\Omega_2 = \mathbb{R}^3 \backslash \bar{\Omega}_1$ with dielectric constant ε_2 containing an ionic solvent. In this work, the dielectric interface $\Gamma = \bar{\Omega}_1 \cap \bar{\Omega}_2$ is taken to be the molecular surface (or solvent excluded surface, SES).[6,7] In a 1:1 electrolyte with low ionic concentration, the linear PB equation for the electrostatic potential ϕ is

$$-\nabla\cdot(\epsilon(\mathbf{x})\nabla\phi(\mathbf{x})) + \bar{\kappa}^2(\mathbf{x})\phi(\mathbf{x}) = \sum_{k=1}^{N_c} q_k\delta(\mathbf{x}-\mathbf{y}_k), \quad \mathbf{x}\in\mathbb{R}^3, \tag{1}$$

where ε is the dielectric constant, $\bar{\kappa}$ is the inverse modified Debye length in units of Å^{-1}, N_c is the number of atomic point charges representing the biomolecule, $\mathbf{y}_k$ is the position of the kth atom, and q_k is its partial charge in units of elementary charge e_c. The interface conditions on the molecular surface are

$$\phi_1(\mathbf{x}) = \phi_2(\mathbf{x}),\ \epsilon_1\frac{\partial\phi_1(\mathbf{x})}{\partial n} = \epsilon_2\frac{\partial\phi_2(\mathbf{x})}{\partial n}, \quad \mathbf{x}\in\Gamma \tag{2}$$

where ϕ_1, ϕ_2 are the limiting values for $\mathbf{x}\to\Gamma$ from inside and outside Ω_1, respectively, and n denotes the outward normal on Γ. The potential also satisfies the far-field boundary condition, $\phi(\mathbf{x})\to 0$ as $|\mathbf{x}|\to\infty$. In the present work ε, $\bar{\kappa}$ are taken as piecewise constant,

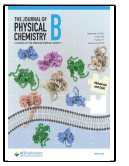
$$\epsilon(\mathbf{x}) = \begin{cases}\epsilon_1, & \mathbf{x}\in\Omega_1,\\ \epsilon_2, & \mathbf{x}\in\Omega_2,\end{cases} \qquad \bar{\kappa}^2(\mathbf{x}) = \begin{cases}0, & \mathbf{x}\in\Omega_1,\\ \dfrac{8\pi N_A e_c^2}{1000 k_B T} I_s, & \mathbf{x}\in\Omega_2,\end{cases} \tag{3}$$

where N_A is Avogadro's number, k_B is Boltzmann's constant, T is absolute temperature, and I_s is molar concentration of the ionic solvent. A key quantity of interest is the electrostatic solvation energy,

$$\Delta G_{solv} = \frac{1}{2}\sum_{k=1}^{N_c} q_k\phi_{reac}(\mathbf{y}_k), \tag{4}$$

where the reaction field potential at a solute atom is the difference between the total potential and the Coulomb potential,

Received: June 30, 2022
Revised: August 28, 2022
Published: September 14, 2022



https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

$$\phi_{\text{reac}}(\mathbf{y}_k) = \lim_{\mathbf{x}\to\mathbf{y}_k}\left(\phi(\mathbf{x}) - \frac{1}{\epsilon_1}\sum_{j=1}^{N_c}\frac{q_j}{4\pi|\mathbf{x}-\mathbf{y}_j|}\right) \tag{5}$$

It should be noted that the electrostatic solvation energy in eq 4 is a useful metric for assessing accuracy, but it is not physically relevant for the biomolecular systems considered in this study due to the lack of treatment of many other physical effects (nonpolar, conformational/unfolding, titration, etc.).

METHODS

A variety of methods have been applied to solve the PB equation including finite-difference,[8−20] finite-element,[5,21,22] domain decomposition,[23] and boundary integral[24−33] methods. The present work is concerned with a treecode-accelerated boundary integral method called TABI-PB.[31,34] In most cases, boundary integral PB solvers compute the potential on a triangulation of the molecular surface, and while they benefit from rigorous enforcement of the interface and far-field boundary conditions, they face the challenge of discretizing singular surface integrals and the expense of solving a dense linear system.

The boundary integral form of the PB equation is not unique, and various forms have different condition numbers,[26] but Juffer et al.[27] developed a well-conditioned form that couples the surface potential and its interior normal derivative on the interface,

$$\frac{1}{2}(1+\epsilon)\phi(\mathbf{x}) = \int_\Gamma\left[K_1(\mathbf{x},\mathbf{y})\frac{\partial\phi(\mathbf{y})}{\partial n} + K_2(\mathbf{x},\mathbf{y})\phi(\mathbf{y})\right]dS_\mathbf{y} + S_1(\mathbf{x}) \tag{6a}$$

$$\frac{1}{2}(1+\epsilon^{-1})\frac{\partial\phi(\mathbf{x})}{\partial n} = \int_\Gamma\left[K_3(\mathbf{x},\mathbf{y})\frac{\partial\phi(\mathbf{y})}{\partial n} + K_4(\mathbf{x},\mathbf{y})\phi(\mathbf{y})\right]dS_\mathbf{y} + S_2(\mathbf{x}) \tag{6b}$$

where $\mathbf{x}\in\Gamma$ and $\varepsilon = \varepsilon_2/\varepsilon_1$ is the solvent/solute ratio of dielectric constants. The kernels depend on the Coulomb and screened Coulomb potentials,

$$G_0(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|}, \qquad G_\kappa(\mathbf{x},\mathbf{y}) = \frac{e^{-\kappa|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} \tag{7}$$

with $\kappa^2 = \bar{\kappa}^2/\epsilon_2$, and are given by

$$K_1(\mathbf{x},\mathbf{y}) = G_0(\mathbf{x},\mathbf{y}) - G_\kappa(\mathbf{x},\mathbf{y}), \quad K_4(\mathbf{x},\mathbf{y}) = -\frac{\partial^2 K_1(\mathbf{x},\mathbf{y})}{\partial n_\mathbf{x}\partial n_\mathbf{y}} \tag{8a}$$

$$K_2(\mathbf{x},\mathbf{y}) = \frac{\partial H_2(\mathbf{x},\mathbf{y})}{\partial n_\mathbf{y}}, \quad H_2(\mathbf{x},\mathbf{y}) = \epsilon G_\kappa(\mathbf{x},\mathbf{y}) - G_0(\mathbf{x},\mathbf{y}) \tag{8b}$$

$$K_3(\mathbf{x},\mathbf{y}) = \frac{\partial H_3(\mathbf{x},\mathbf{y})}{\partial n_\mathbf{x}}, \quad H_3(\mathbf{x},\mathbf{y}) = G_0(\mathbf{x},\mathbf{y}) - \epsilon^{-1}G_\kappa(\mathbf{x},\mathbf{y}) \tag{8c}$$

The source terms in eq 6 depend on the solute atoms,

$$S_1(\mathbf{x}) = \frac{1}{\epsilon_1}\sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad S_2(\mathbf{x}) = \frac{1}{\epsilon_1}\sum_{k=1}^{N_c} q_k\frac{\partial G_0(\mathbf{x},\mathbf{y}_k)}{\partial n_\mathbf{x}}. \tag{9}$$

Our earlier TABI-PB 1.0 solver[31] triangulated the molecular surface using MSMS[35,36] and computed the surface integrals by collocation at the triangle centroids $\mathbf{x}_i$, $i = 1{:}N$. This yields a linear system for the surface potential and its normal derivative at the centroids,

$$\frac{1}{2}(1+\epsilon)\phi(\mathbf{x}_i) = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_1(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi(\mathbf{x}_j)}{\partial n} + K_2(\mathbf{x}_i,\mathbf{x}_j)\phi(\mathbf{x}_j)\right]A_j + S_1(\mathbf{x}_i), \tag{10a}$$

$$\frac{1}{2}(1+\epsilon^{-1})\frac{\partial\phi(\mathbf{x}_i)}{\partial n} = \sum_{\substack{j=1\\ j\neq i}}^{N}\left[K_3(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi(\mathbf{x}_j)}{\partial n} + K_4(\mathbf{x}_i,\mathbf{x}_j)\phi(\mathbf{x}_j)\right]A_j + S_2(\mathbf{x}_i), \tag{10b}$$

where $i = 1{:}N$, and A_j is the area of the jth triangle. The linear system was solved by GMRES[37] and at each step the matrix-vector product was computed using a Taylor treecode.[38] In this context, the electrostatic solvation energy is

$$\Delta G_{\text{solv}} = \frac{1}{2}\sum_{k=1}^{N_c} q_k\sum_{j=1}^{N}\left[K_1(\mathbf{y}_k,\mathbf{x}_j)\frac{\partial\phi(\mathbf{x}_j)}{\partial n} + K_2(\mathbf{y}_k,\mathbf{x}_j)\phi(\mathbf{x}_j)\right]A_j, \tag{11}$$

where the singularity in eq 5 has been analytically canceled and the reaction field potential $\phi_{\text{reac}}(\mathbf{y}_k)$ is given by the inner sum in eq 11.

The present article describes an improved boundary integral solver called TABI-PB 2.0 utilizing the NanoShaper surface triangulation code,[39−41] node-patch discretization,[30] a block preconditioner,[42] and a GPU-accelerated fast multipole method called BLDTT based on barycentric Lagrange interpolation and dual tree traversal.[43] The following sections describe these techniques.

Surface Triangulation and Preconditioning. In this work, the dielectric interface separating the solute and solvent is taken to be the molecular surface or solvent-excluded surface (SES).[6,7] In boundary integral PB simulations, the quality of the surface triangulation critically affects the condition number of the linear system, and an ill-conditioned system requires more GMRES iterations to converge. TABI-PB 1.0 utilized the MSMS triangulation code which creates an analytical surface representation and generates a triangulation by fitting predefined triangulated patches.[35,36] In some cases, MSMS can produce triangles of exceedingly small area and large aspect ratio leading to a poorly conditioned linear system, but this is alleviated by block preconditioning.[42] More recently the NanoShaper triangulation code was developed which builds a patch description of the surface and employs ray-casting and a marching cubes algorithm to obtain the triangulation.[39−41] A comparison of MSMS and NanoShaper for a set of 38 biomolecules showed that the two codes yield comparable values for the SES surface area and electrostatic solvation energy,

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

but NanoShaper computations were more efficient and reliable, especially when parameters were set to produce highly resolved triangulations.[44]

Boundary Integral Discretization. Given a triangulation of the molecular surface, TABI-PB 1.0 discretized the surface integrals by centroid collocation to obtain the potential and its normal derivative at the triangle centroids. The node-patch scheme is an alternative that yields the surface potential and its normal derivative at the triangle vertices.[30] In the node-patch scheme, a node is a triangle vertex and a patch is a portion of the surface surrounding the vertex. Figure 1 depicts the two schemes

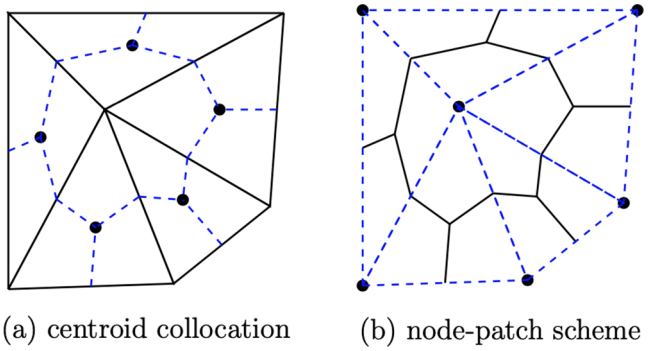


Figure 1. Boundary integral discretization, five triangles meeting at a vertex: (a) the centroid collocation uses triangle centroids, and the quadrature weight is a triangle area; (b) the node-patch scheme uses triangle vertices, and the quadrature weight is a patch area formed by summing one-third of surrounding triangle areas.[30]

for five triangles meeting at a vertex; the quadrature weights for centroid collocation are the triangle areas, while the quadrature weights for the node-patch scheme are the patch areas formed by summing one-third of the surrounding triangle areas. The linear system has the same form as in eq 10, but since a surface triangulation has roughly half as many vertices as centroids, the node-patch scheme reduces the system size while maintaining the same level of surface resolution.[30]

Matrix-Vector Product. Solving the linear system in eq 10 by GMRES requires computing matrix-vector products in the form of N-body potentials,

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} K(\mathbf{x}_i, \mathbf{y}_j) q_j, \qquad i = 1{:}\,N, \tag{12}$$

where $\mathbf{x}_i$, $\mathbf{y}_j$ are triangle centroids or vertices and q_j is a charge associated with $\mathbf{y}_j$. The $\mathbf{x}_i$ are considered as target particles and the $\mathbf{y}_j$ as source particles; the two sets are allowed to be different as required in computing the source terms in eq 9 and solvation energy in eq 11, but for simplicity below, we consider them as two copies of the same set.

The run time for computing the sums in eq 12 by direct summation scales like $O(N^2)$, but TABI-PB 1.0 used a Taylor treecode with run time $O(N \log N)$,[31,38] while TABI-PB 2.0 uses the recently developed barycentric Lagrange dual tree traversal (BLDTT) fast multipole method with run time $O(N)$.[43,45,46] In the BLDTT, each set of particles is divided into a hierarchical tree of cuboid clusters (rectangular boxes with sides parallel to the Cartesian axes); hence there are two trees, the target tree with clusters C_t and the source tree with clusters C_s. The computed target potential $\phi(\mathbf{x}_i)$ has contributions from (1) direct interactions with nearby source clusters, and (2) approximate interactions with well-separated source clusters obtained by applying barycentric Lagrange interpolation[47] to the kernel $K(\mathbf{x}_i, \mathbf{y}_j)$.

Barycentric Lagrange Interpolation. Let $f(x)$ be a given function for $x \in [-1, 1]$ and consider interpolation at Chebyshev points $s_k = \cos(\pi k/n)$, $k = 0{:}n$, where $n \geq 1$ is an integer. The barycentric Lagrange form of the interpolating polynomial is

$$p(x) = \sum_{k=0}^{n} f(s_k) L_k(x), \qquad L_k(x) = \frac{\dfrac{w_k}{x - s_k}}{\sum_{k'=0}^{n} \dfrac{w_{k'}}{x - s_{k'}}}, \tag{13}$$

where the barycentric weights are

$$w_k = (-1)^k \delta_k, \qquad \delta_k = \begin{cases} 1/2 & \text{if } k = 0 \text{ or } k = n, \\ 1 & \text{if } k = 1{:}\, n-1. \end{cases} \tag{14}$$

Barycentric Lagrange interpolation is efficient and stable, and it is scale-invariant in that to interpolate on an interval $[a, b]$ other than $[-1, 1]$, the Chebyshev points are linearly mapped to the interval, but the weights w_k stay the same.[47] Barycentric Lagrange interpolation extends in a straightforward way to functions defined on cuboids in 3D, where the interpolation points form a tensor product grid of Chebyshev points in each Cartesian coordinate.

In the BLDTT, barycentric Lagrange interpolation is applied to the kernel $K(\mathbf{x}, \mathbf{y})$, utilizing Chebyshev grid points $\mathbf{t}_l = (t_{l_1}, t_{l_2}, t_{l_3})$ in a target cluster and $\mathbf{s}_\mathbf{k} = (s_{k_1}, s_{k_2}, s_{k_3})$ in a source cluster. The Chebyshev grid points t_l, $\mathbf{s}_\mathbf{k}$ are considered to be *proxy particles* because they replace the actual particles $\mathbf{x}_i$, $\mathbf{y}_j$. There are four types of clusters as depicted in Figure 2, (a) target

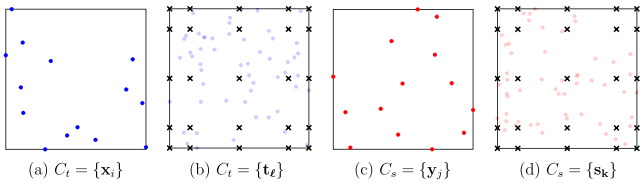


Figure 2. Four types of clusters. Blue/red dots are target/source particles, crosses are proxy particles. Key: (a) target cluster, (b) proxy target cluster, (c) source cluster, and (d) proxy source cluster. Numerous light blue/red particles in (b) and (d) are replaced by proxy particles.

cluster $C_t = \{\mathbf{x}_i\}$, (b) proxy target cluster $C_t = \{\boldsymbol{t}_l\}$, (c) source cluster $C_s = \{\mathbf{y}_j\}$, and (d) proxy source cluster $C_s = \{\mathbf{s}_\mathbf{k}\}$, where the filled dots are particles (targets/sources are blue/red) and the crosses are proxy particles. Proxy clusters are used when the actual clusters contain sufficiently many particles; hence the numerous light blue/red particles in (b) and (d) are replaced by a smaller set of proxy particles.

Cluster Interactions. Figure 2 showed there are two types of target clusters C_t (a, b) and two types of source clusters C_s (c, d); hence, the BLDTT allows four types of cluster interactions,

(1) particle−particle (PP): particles in C_t interact with particles in C_s;
(2) particle−cluster (PC): particles in C_t interact with proxy particles in C_s;
(3) cluster−particle (CP): proxy particles in C_t interact with particles in C_s;
(4) cluster−cluster (CC): proxy particles in C_t interact with proxy particles in C_s.

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

Table 1. Four Ways of Evaluating $K(x_i, y_j)$: (1) Particle−Particle (PP), (2) Particle−Cluster (PC), (3) Cluster−Particle (CP), and (4) Cluster−Cluster (CC)[a]

PP, direct	PC, interpolate in y	CP, interpolate in x	CC, interpolate in x and y
$K(x_i, y_j)$	$\sum_{k=0}^{n} K(x_i, s_k)L_k(y_j)$	$\sum_{l=0}^{n} L_l(x_i)K(t_l, y_j)$	$\sum_{k=0}^{n}\sum_{l=0}^{n} L_l(x_i)K(t_l, s_k)L_k(y_j)$
potentials $\phi(x_i)$		proxy potentials $\phi(t_l)$	
$\sum_{y_j \in C_s} K(x_i, y_j)q_j$	$\sum_{k=0}^{n} K(x_i, s_k)\hat{q}_k$	$\sum_{y_j \in C_s} K(t_l, y_j)q_j$	$\sum_{k=0}^{n} K(t_l, s_k)\hat{q}_k$

[a]PP and PC yield potentials $\phi(x_i)$, CP and CC yield proxy potentials $\phi(t_l)$, and 1D notation used for simplicity.

Table 1 shows the corresponding four ways of evaluating the kernel $K(\mathbf{x}_i, \mathbf{y}_j)$; (1) PP uses direct evaluation, (2) PC interpolates with respect to $\mathbf{y}$, (3) CP interpolates with respect to $\mathbf{x}$, and (4) CC interpolates with respect to $\mathbf{x}$ and $\mathbf{y}$, where in each case the interpolation is done using the barycentric Lagrange form in eq 13. Note that Table 1 uses 1D notation for simplicity and the extension to 3D is straightforward. Table 1 also indicates that PP and PC yield potentials $\phi(x_i)$, while CP and CC yield proxy potentials $\phi(t_l)$. In 3D, the *proxy charges* arising in PC and CC are defined by

$$\hat{q}_{\mathbf{k}} = \sum_{\mathbf{y}_j \in C_s} L_{k_1}(y_{j1})L_{k_2}(y_{j2})L_{k_3}(y_{j3})q_j. \tag{15}$$

The proxy charges are associated with proxy source particles, and they are computed by an upward pass similar to that in the FMM,[48] although here it is adapted to polynomial interpolation.[43]

Note that in all cases the expressions for the potentials and proxy potentials in Table 1 have a direct sum form requiring only kernel evaluations for suitable target and source particles and their proxies; this enables an efficient GPU implementation without thread divergence as discussed below, but first, we explain the dual tree traversal that determines the interaction list for target and source clusters.[49,50]

Dual Tree Traversal. Before the traversal starts, two sets of potentials are initialized to zero, potentials $\phi(\mathbf{x}_i)$ at the target particles and proxy potentials $\phi(\mathbf{t}_l)$ at the proxy target particles. During the traversal, potentials $\phi(\mathbf{x}_i)$ are incremented due to PP and PC interactions, and proxy potentials $\phi(\mathbf{t}_l)$ are incremented due to CP and CC interactions. Following the traversal, the proxy potentials $\phi(\mathbf{t}_l)$ are interpolated to the target particles $\mathbf{x}_i$ and combined with the potentials $\phi(\mathbf{x}_i)$ in a downward pass as in the FMM.[43,48]

The dual tree traversal uses a recursive procedure taking a target cluster C_t and source cluster C_s as input, starting from the root clusters of the target and source trees. The clusters are considered to be *well-separated* if $(r_t + r_s)/R < \theta$, where r_t, r_s are the cluster radii, R is their center−center distance, and θ is the user-specified multipole acceptance criterion (MAC) parameter.

If C_t and C_s are well-separated, they interact in one of four ways depending on the number of particles they contain ($|C_t|$, $|C_s|$) in comparison with the number of proxy particles ($n_p = (n+1)^3$). If $|C_t| > n_p$ and $|C_s| > n_p$, then CC proxy potentials are computed; else if $|C_t| > n_p$ and $|C_s| \le n_p$, then CP proxy potentials are computed; else if $|C_t| \le n_p$ and $|C_s| > n_p$, then PC potentials are computed; else $|C_t| \le n_p$ and $|C_s| \le n_p$, and then PP potentials are computed.

If C_t and C_s are not well-separated, the traversal proceeds as follows. If C_t and C_s are both leaves, they interact directly. If C_t is not a leaf but C_s is a leaf, then C_s interacts recursively with the children of C_t; conversely, if C_t is a leaf but C_s is not a leaf, then C_t interacts recursively with the children of C_s. Otherwise, C_t and C_s are not leaves, and the smaller cluster interacts recursively with the children of the larger cluster.

Application to TABI-PB 2.0. The main cost in computing the matrix-vector product required in GMRES consists of evaluating the sums in eq 10, and we shall write them in a more concise form better suited for the BLDTT. Let (a_i) denote the input vector of the sums, where entries $i = 1{:}N$ are the potentials $\phi(\mathbf{x}_i)$ and entries $i = (N+1){:}2N$ are the normal derivatives $\partial_n\phi(\mathbf{x}_i)$. Then (b_i) is the output vector of the sums, where

$$b_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} [K_1(\mathbf{x}_i, \mathbf{x}_j)a_{N+j} + \mathbf{n}(\mathbf{x}_j)\cdot\nabla_{\mathbf{y}}H_2(\mathbf{x}_i, \mathbf{x}_j)a_j]A_j, \tag{16a}$$

$$b_{N+i} = \sum_{\substack{j=1 \\ j\neq i}}^{N} [\mathbf{n}(\mathbf{x}_i)\cdot\nabla_{\mathbf{x}}H_3(\mathbf{x}_i, \mathbf{x}_j)a_{N+j} - \mathbf{n}(\mathbf{x}_i)\mathbf{n}(\mathbf{x}_j) : \nabla^2_{\mathbf{xy}}K_1(\mathbf{x}_i, \mathbf{x}_j)a_j)]A_j. \tag{16b}$$

Then the desired concise form of the sums is

$$b_i = p_0(i)\sum_{j=1}^{N} V_0(i, j), \qquad b_{N+i} = \sum_{m=1}^{3} p_m(i)\sum_{j=1}^{N} V_m(i, j), \tag{17}$$

where Table 2 defines new target charges $p_m(i)$, source charges $q_m(j)$, and potentials $V_m(i, j)$, for $m = 0{:}3$.

At each step of GMRES, the BLDTT computes the four sums in eq 17 involving $V_m(i, j)$, $m = 0{:}3$, where the target and source trees are built on the triangle vertices as required by node-patch discretization. The source terms $S_1(\mathbf{x}_i)$, $S_2(\mathbf{x}_i)$ in eq 9 are also computed using the BLDTT, where the target particles are the triangle vertices, $\mathbf{x}_i$, $i = 1{:}N$, and the source particles are the atomic point charges representing the solute, $\mathbf{y}_k$, $k = 1{:}N_c$. The solvation energy ΔG_{solv} in eq 11 is computed similarly except that the target and source particles are reversed. These computations rely on the capability of the BLDTT to handle disjoint sets of target and source particles.[43,46]

GPU Implementation. In TABI-PB 2.0, the matrix−vector product in each step of GMRES is computed using a GPU implementation of the BLDTT, where the compute kernels are generated by OpenACC directives and compiled with the NVIDIA HPC nvc++ compiler. The four most important GPU

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

Table 2. Definitions for the Concise Form of Sums in Equation 17 Required for the Matrix–Vector Product, Target Charges $p_m(i)$, Source Charges $q_m(j)$, and Potentials $V_m(i, j)$[a]

m	$p_m(i)$	$q_m(j)$	$V_m(i, j)$
0	1	$a_{N+j}A_j$	$q_0(j)K_1 + \sum_{m=1}^{3} q_m(j)n_m(\mathbf{x}_j)\partial_{x_{jm}}H_2$
1	$n_1(\mathbf{x}_i)$	$n_1(\mathbf{x}_j)a_jA_j$	$n_1(\mathbf{x}_i)[q_0(j)\partial_{x_{i1}}H_3 - \sum_{m=1}^{3} q_m(j)n_m(\mathbf{x}_j)\partial_{x_{i1}}\partial_{x_{jm}}K_1]$
2	$n_2(\mathbf{x}_i)$	$n_2(\mathbf{x}_j)a_jA_j$	$n_2(\mathbf{x}_i)[q_0(j)\partial_{x_{i2}}H_3 - \sum_{m=1}^{3} q_m(j)n_m(\mathbf{x}_j)\partial_{x_{i2}}\partial_{x_{jm}}K_1]$
3	$n_3(\mathbf{x}_i)$	$n_3(\mathbf{x}_j)a_jA_j$	$n_3(\mathbf{x}_i)[q_0(j)\partial_{x_{i3}}H_3 - \sum_{m=1}^{3} q_m(j)n_m(\mathbf{x}_j)\partial_{x_{i3}}\partial_{x_{jm}}K_1]$

[a],Kernels K_1, H_2, and H_3 are evaluated at $(\mathbf{x}_i, \mathbf{x}_j)$ with definitions in eq 8).

compute kernels correspond to PP, PC, CP, and CC interactions between a target cluster C_t and source cluster C_s on the interaction list. The compute kernels evaluate the potentials and proxy potentials defined in Table 1 as required for the subsequent downward pass. Each interaction launches a compute kernel asynchronously, and further computation is blocked until all compute kernels for a given GMRES iteration have completed.

The four compute kernels have similar structure, with an outer loop over target particles $\mathbf{x}_i$ or proxy target particles $\mathbf{t}_l$ in C_t, and an inner loop over source particles $\mathbf{y}_j$ or proxy source particles $\mathbf{s}_k$ in C_s. In all cases the expressions for the potentials and proxy potentials in Table 1 have a direct sum form requiring only kernel evaluations for suitable target and source particles and their proxies; as a result, the inner loop iterations are independent of each other and can be computed concurrently without thread divergence. The outer loop maps to the OpenACC `gang` construct and the inner loop maps to the `vector` construct, where, roughly speaking, a `gang` member corresponds to a CUDA thread block and a `vector` member corresponds to an individual CUDA thread. Note that in addition to the matrix–vector product, the proxy charges, PB source terms, and solvation energy were also computed using the GPU-accelerated BLDTT. Further details of the GPU implementation are reported elsewhere.[43,51]

RESULTS AND DISCUSSION

We consider the three solutes in Table 3, protein 1A63 and two viral capsids (Zika, H1N1), showing their PDB ID if available,

Table 3. Solute Examples

	PDB ID	N_c atoms	bounding box
protein	1A63	2065	26 Å × 38 Å × 34 Å
Zika capsid[53]	6CO8	1 576 628	467 Å × 460 Å × 465 Å
H1N1 capsid[54]	–	14 442 610	1160 Å × 1153 Å × 1237 Å

number of atoms, and bounding box dimensions. Note that the bounding box dimensions are given here only to indicate the relative sizes of the solutes and are not used in the TABI-PB 2.0 solver. The PQR files were generated using PDB 2PQR[52] with the CHARMM force field. The dielectric constants were $\varepsilon_1 = 1$ for protein 1A63, $\varepsilon_1 = 4$ for the viral capsids, and $\varepsilon_2 = 80$ for the solvent. The ionic concentration was $I_s = 150$ mM. The GMRES tolerance was 1×10^{-4} with 10 iterations between restarts.

The TABI-PB 2.0 code was written in C++ and compiled with the NVIDIA HPC nvc++ compiler using the `-O3` optimization flag. The computations were done on SDSC Expanse using one CPU core of a 2.25 GHz AMD EPYC 7742 processor and one NVIDIA V100 GPU. Several operations are done on the CPU including surface triangulation by NanoShaper, building the tree data structure, and computing interaction lists by dual tree traversal, while the cluster interactions are done either on the GPU or on the CPU for comparison as indicated below. We compare two boundary integral discretization schemes (CC = centroid collocation, NP = node-patch) and three matrix-vector product methods (DS = direct summation, TTC = Taylor treecode, and BLDTT = barycentric Lagrange dual tree traversal). In principle, the code should be able to run on an AMD GPU using the g++ compiler with OpenACC from the GCC, but that configuration has not yet been tested. TABI-PB 2.0 can also run on a single CPU compute node with OpenMP threading using any standard C++ compiler supported on Mac, Linux, or Windows systems.

1A63 CPU Performance. This section presents results for protein 1A63 running on one CPU core. Three versions of the code are compared, one using centroid collocation and the Taylor treecode as in TABI-PB 1.0 (CC-TTC), and two using the node-patch scheme with the Taylor treecode or the BLDTT (NP-TTC, NP-BLDTT). The TTC computations used MAC $\theta = 0.8$, expansion order $p = 2$, maximum leaf size $N_0 = 500$, and the BLDTT computations used MAC $\theta = 0.8$, interpolation degree $n = 2$, maximum leaf size $N_0 = 50$; these values ensure that the error in computing the matrix–vector product is less than the CC/NP discretization error. Table 4 gives the NanoShaper

Table 4. Protein 1A63, Surface Triangulation with NanoShaper Scale s, System Size N, Number of Faces in Centroid Collocation (CC), and the Number of Vertices in the Node-Patch Scheme (NP)

scale s	CC N (faces)	NP N (vertices)
1.0	20 688	10 350
1.5	47 496	23 754
2.0	84 512	42 260
2.5	132 612	66 312
3.0	191 124	95 568
3.5	260 280	130 146
4.0	340 116	170 064

scale s used to triangulate the molecular surface and the corresponding system size N, which is the number of faces in centroid collocation or the number of vertices in node-patch. As expected, for a given triangulation, the node-patch scheme uses roughly half as many degrees of freedom as centroid collocation.

Figure 3 presents (a) solvation energy ΔG_{solv} (kcal/mol) and (b) ΔG_{solv} relative error (%) versus $1/N$, where N is the system size from Table 4. Note that part a is plotted in linear scale and part b is plotted in logarithmic scale. The reference value for computing the error is obtained by extrapolation to the limit $N \to \infty$; the procedure will be explained below in the context of the Zika capsid. Figure 3a shows that, for all three versions of the code, the computed ΔG_{solv} converges smoothly as the surface triangulation is refined. Figure 3b indicates that the three versions converge roughly at the rate $O(N^{-1})$, although for a given system size N, the node-patch is more accurate than

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104–7113

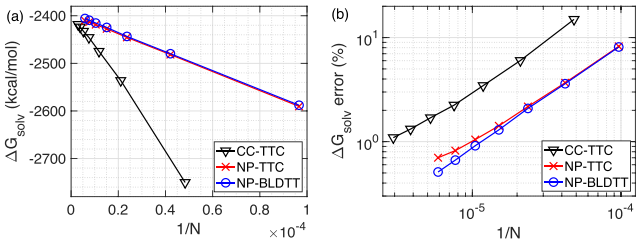


Figure 3. Protein 1A63: (a) solvation energy ΔG_{solv} (kcal/mol) and (b) ΔG_{solv} relative error (%) versus $1/N$ for system size N from Table 4, comparing CC-TTC, NP-TTC, and NP-BLDTT computations on one CPU core.

centroid collocation, and BLDTT is slightly more accurate than TTC. The latter observation can be understood by noting that the Taylor approximation error in the TTC is nonuniform; in particular, it is larger for target particles closer to the boundary of the source cluster, but this effect is absent in the BLDTT since Chebyshev interpolation controls the approximation error uniformly. The accuracy of the TTC degrades slightly as the triangulation is refined because in that case there is more opportunity for target particles to be close to source clusters.

Figure 4a presents run time (s) versus system size N, showing that NP-BLDTT is faster than NP-TTC and CC-TTC (except

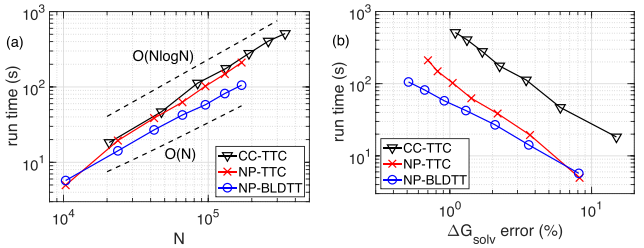


Figure 4. Protein 1A63, run time (s) versus (a) system size N from Table 4 and (b) ΔG_{solv} relative error (%), comparing CC-TTC, NP-TTC, and NP-BLDTT computations on one CPU core.

for the smallest system size N = 1e4 where NP-TTC is slightly faster). The run time for NP-TTC and CC-TTC scales approximately like $O(N \log N)$ as expected for a treecode, while the run time for NP-BLDTT scales closer to $O(N)$ as expected for a fast multipole method[43,48] Figure 4b presents run time (s) versus ΔG_{solv} relative error (%). The gap between CC-TTC and the two node-patch codes is consistent with the error results in Figure 3b. The results show that NP-BLDTT achieves smaller error with less run time than either NP-TTC or CC-TTC (except for the largest error data point where NP-TTC is slightly faster).

Table 5 displays the results in detail, where the top portion used centroid collocation and the bottom portion used node-patch, and the matrix–vector product was computed by DS, TTC, or BLDTT. Column 1 gives the NanoShaper scale s and column 2 gives the system size N (number of faces for CC, number of vertices for NP). Columns 3–5 give the solvation energy ΔG_{solv} (kcal/mol), where the row $N = \infty$ is the reference value obtained by extrapolation as described below. Note that the reference values for CC and NP agree to within 1.5 kcal/mol, which is less than 0.07%. Columns 6–8 present the ΔG_{solv} relative error (%). Column 6 (DS) is the boundary integral discretization error measuring the deviation between the computed ΔG_{solv} and the reference value, showing that for a given scale, NP has significantly smaller discretization error than CC. Columns 7–8 (TTC, BLDTT) give the fast summation approximation error measuring the deviation between the ΔG_{solv} values in columns 4–5 and the value in column 3; the BLDTT approximation error is less than the TTC approximation error, which in turn is less than the DS discretization error. Columns 9–11 give the run time (s) showing that NP is faster than CC, while BLDTT is faster than TTC, which in turn is faster than DS.

1A63 GPU Performance. The following results also pertain to protein 1A63 with the same BLDTT parameters and errors as in Table 5. Figure 5 plots the NP-BLDTT run time (s) versus system size N on one CPU core and on one GPU, where dashed lines include the run time for surface triangulation which is done on the CPU, and solid lines exclude the triangulation time. Table 6 presents the run time (s) plotted in Figure 5 together with the speedup of the GPU over the CPU. For the largest scale s = 4, NP-BLDTT computes ΔG_{solv} with 0.5% error in 6.81 s on the GPU; moreover the code runs 15 times faster on the GPU than on the CPU, and the speedup doubles if the triangulation run time on the CPU is excluded.

Figure 6 shows the fraction of NP-BLDTT run time by component versus scale s on (a) one CPU, and (b) one GPU. The components are NanoShaper surface triangulation (blue), source terms S_1, S_2 (orange), upward pass for proxy charges $\hat{q}_{\mathbf{k}}$ (yellow), downward pass to interpolate proxy potentials $\phi(t_l)$ to potentials $\phi(\mathbf{x}_i)$ (purple), PP, PC, CP, CC cluster interactions (green), computing ΔG_{solv} (light blue), and other (burgundy) including tree building and constructing interaction lists. In the CPU computation (a), the run time fraction for each component varies little with scale s, and most of the run time is due to cluster interactions. In the GPU computation (b), the largest fraction of run time is due to triangulation (which is actually done on the CPU), followed by cluster interactions (except at the smallest scales s = 1, 1.5); this implies that a future GPU implementation of NanoShaper will yield a sizable speedup in GPU run time. Note that in the GPU computation (b), the run times for the source terms and ΔG_{solv} are too small to be visible.

Viral Capsids. Table 7 shows results for the Zika and H1N1 viral capsids whose data were given in Table 3. The computations were done using NP-BLDTT on one GPU with parameters MAC θ = 0.8, degree n = 3, maximum leaf size N_0 = 500. Columns 1–2 give the NanoShaper scale s and system size N. The Zika computations used three scales s = 1, 1.5, 2, and since the H1N1 capsid has larger dimensions only a single scale s = 0.5 is presented because larger scales did not yield a valid triangulation. Columns 3–4 give the solvation energy ΔG_{solv} (kcal/mol) and its relative error (%), where the error is computed with respect to the reference value in the row $N = \infty$; the calculation of the reference value is described below. Column 5 gives two values for the run time (s), where t_1 includes the triangulation time done on the CPU and t_2 excludes this time. Column 6 gives the number of GMRES iterations and column 7 gives the peak memory (GB).

Among the Zika results, the row $N = \infty$ gives the ΔG_{solv} reference value for computing the error, which was obtained by extrapolation as follows.[31] Let $\Delta G_{solv}(N)$ be the computed value of the solvation energy for a given system size N and suppose it varies smoothly,

$$\Delta G_{solv}(N) \sim \alpha_0 + \frac{\alpha_1}{N} + \frac{\alpha_2}{N^2} + \cdots \tag{18}$$

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104–7113

Table 5. Protein 1A63, Matrix−Vector Product by DS, TTC, BLDTT, Discretization by Centroid Collocation (CC), Node-Patch (NP)[a]

		ΔG_{solv} (kcal/mol)			ΔG_{solv} error (%)			run time (s)		
s	N	DS	TTC	BLDTT	DS	TTC	BLDTT	DS	TTC	BLDTT
				centroid collocation (CC)						
1.0	20 688	−2749.0	−2750.2	–	14.92	0.046	–	158	18	–
1.5	47 496	−2533.3	−2536.1	–	5.91	0.109	–	849	47	–
2.0	84 512	−2471.7	−2474.8	–	3.33	0.122	–	3 381	112	–
2.5	132 612	−2442.2	−2445.7	–	2.10	0.144	–	7 517	176	–
3.0	191 124	−2427.1	−2432.6	–	1.47	0.225	–	16 706	279	–
3.5	260 280	−2417.7	−2423.6	–	1.07	0.245	–	32 605	406	–
4.0	340 116	−2411.6	−2418.2	–	0.82	0.273	–	55 419	511	–
	∞	−2392.0								
				node-patch (NP)						
1.0	10 350	−2587.7	−2590.4	−2588.0	8.11	0.106	0.010	29	5	6
1.5	23 754	−2479.6	−2481.6	−2479.8	3.60	0.078	0.005	181	19	14
2.0	42 260	−2443.1	−2445.5	−2443.4	2.07	0.097	0.011	610	39	27
2.5	66 312	−2424.4	−2427.4	−2424.6	1.29	0.125	0.007	1 611	63	42
3.0	95 568	−2415.1	−2418.6	−2415.4	0.90	0.142	0.010	3 359	102	58
3.5	130 146	−2409.3	−2413.1	−2409.4	0.66	0.161	0.006	6 659	148	82
4.0	170 064	−2405.5	−2410.3	−2405.7	0.50	0.199	0.010	11 335	211	106
	∞	−2393.5								

[a]Column 1: NanoShaper scale s. Column 2: system size N. Columns 3−5: solvation energy ΔG_{solv} (kcal/mol), reference value in row $N = \infty$ computed by extrapolation. Columns 6−8: ΔG_{solv} relative error (%), where column 6 is discretization error and columns 7 and 8 are fast summation approximation error. Columns 9−11: run time (s). TTC uses $\theta = 0.8$, $p = 2$, and $N_0 = 500$. BLDTT uses $\theta = 0.8$, $n = 2$, and $N_0 = 50$, computations on one CPU core.

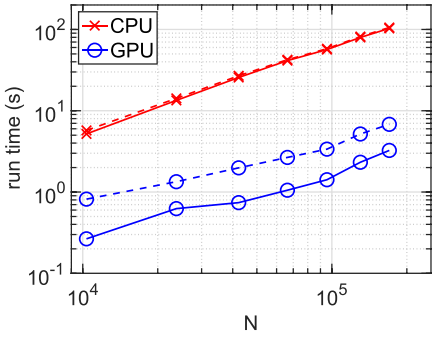


Figure 5. Protein 1A63, NP-BLDTT run time (s) versus system size N, computations on one CPU core (×) and on one GPU (○). Dashed lines include triangulation run time, and solid lines exclude triangulation run time, NP-BLDTT parameters and errors as in Table 5.

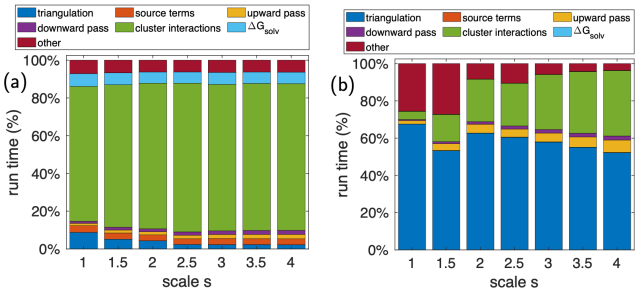


Figure 6. Protein 1A63, fraction of NP-BLDTT run time by component versus scale s, computations on (a) one CPU core, (b) one GPU. Components are NanoShaper surface triangulation (blue), source terms S_1, S_2 (orange), upward pass (yellow), downward pass (purple), cluster interactions for matrix-vector product (green), computing ΔG_{solv} (light blue), and other (burgundy) including tree building and constructing interaction lists.

where the coefficients $\alpha_{0,1,2}$ are obtained by fitting to the three values of N in Table 7. Then α_0 is an estimate of the exact ΔG_{solv} in the limit $N \to \infty$; this value appears in Table 7 in the row $N =$

Table 6. Protein 1A63, NP-BLDTT Run Time (s), (a) Including Triangulation and (b) Excluding Triangulation[a]

			(a) run time (s) including triangulation			(b) run time (s) excluding triangulation		
s	N	error (%)	CPU	GPU	speedup	CPU	GPU	speedup
1.0	10 350	8.11	5.73	0.82	7.0	5.19	0.27	19.5
1.5	23 754	3.60	14.19	1.34	10.6	13.40	0.63	21.5
2.0	42 260	2.07	26.90	1.98	13.6	25.65	0.74	34.7
2.5	66 312	1.29	42.44	2.66	15.9	41.38	1.05	39.3
3.0	95 568	0.90	57.97	3.37	17.2	56.56	1.42	39.9
3.5	130 146	0.66	81.77	5.17	15.8	79.76	2.32	34.3
4.0	170 064	0.50	105.61	6.81	15.5	103.09	3.25	31.7

[a]Scale s, system size N, ΔG_{solv} relative error (%), computations on one CPU core and on one GPU, speedup of GPU over CPU, NP-BLDTT parameters as in Table 5.

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

Table 7. Zika and H1N1 Viral Capsids, NanoShaper Scale s, System Size N, Solvation Energy ΔG_{solv} (kcal/mol), Relative Error (%), Run Time (s), t_1 Includes Triangulation, t_2 Excludes Triangulation, Number of GMRES Iterations (iter), and Peak Memory (GB)[a]

				run time (s)			
s	N	ΔG_{solv} (kcal/mol)	error (%)	t_1	t_2	iter	mem (GB)
			Zika[53]				
1.0	5 109 760	−117 632.1	5.7	394	223	16	6.8
1.5	11 653 254	−113 707.4	2.2	1124	790	26	9.6
2.0	20 832 704	−112 565.8	1.2	1685	1096	20	15.4
	∞	−111 270.6					
			H1N1[54]				
0.5	12 313 670	−31 424 958.2		1112	555	14	37.4

[a]Row $N = \infty$ is extrapolated ΔG_{solv} reference value for computing error, computations by NP-BLDTT on one NVIDIA V100 GPU.

∞, and it is the reference value for computing the error. Figure 7 illustrates the extrapolation procedure by plotting the right side of eq 18 as a function of inverse system size $1/N$.

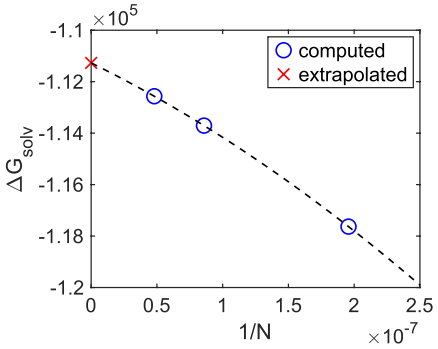


Figure 7. Zika viral capsid,[53] computed ΔG_{solv} (○) versus inverse system size $1/N$, dashed line plots right side of eq 18 with coefficients α_i fit to data in Table 7, extrapolated value $\Delta G_{solv} = -111\,270.6$ (×) is the estimated exact value in row $N = \infty$ in Table 7.

Returning to Table 7, for example with scale $s = 1.5$ and system size $N = 11\,653\,254$, the computed solvation energy $\Delta G_{solv} = -113\,707.4$ kcal/mol has error 2.2%, the run time including triangulation was 1124 s, and the calculation required 26 GMRES iterations and 9.6 GB of memory. These results compare favorably with another recent computation of the Zika solvation energy on a dual 20-core CPU node using a Galerkin boundary integral discretization and an alternative version of the fast multipole method.[55]

The H1N1 viral capsid has about nine times as many atoms as the Zika capsid and the ΔG_{solv} is about 30 times lower. As mentioned, valid H1N1 triangulations could not be obtained for larger scale s, so we are unable to estimate the error at this time. Figure 8 shows the Zika surface potential at scale $s = 1.5$ and the H1N1 surface potential at scale $s = 0.5$; note that the spatial dimensions in these plots are in arbitrary units.

CONCLUSIONS

This work described an improved version of the treecode-accelerated boundary integral Poisson-Boltzmann solver called TABI-PB 2.0. The code computes the electrostatic potential on the molecular surface of a solvated biomolecule and further processing yields the electrostatic solvation energy ΔG_{solv}. The new implementation utilizes the NanoShaper surface triangulation code, node-patch boundary integral discretization, a block preconditioner, and a fast multipole method based on

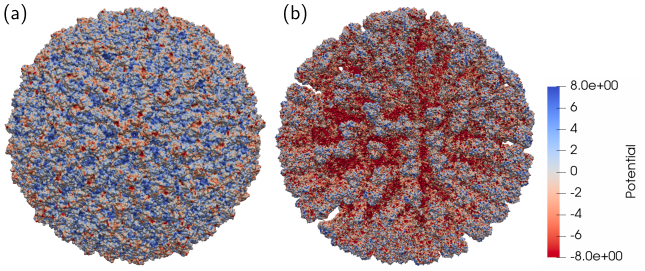


Figure 8. Viral capsid surface potential (kcal/mol/e_c): (a) Zika,[53] NanoShaper scale $s = 1.5$; (b) H1N1,[54] NanoShaper scale $s = 0.5$, spatial dimensions in arbitrary units, and NP-BLDTT computations on one NVIDIA V100 GPU.

barycentric Lagrange interpolation and dual tree traversal (BLDTT). Performance-critical portions of the code were implemented on a GPU including the matrix−vector product in the GMRES solution of the linear system, and computing the proxy charges, PB source terms, and solvation energy. The BLDTT approximations have a direct sum form that facilitates their GPU implementation without thread divergence. Results presented for protein 1A63 (2065 atoms) demonstrate the new code's accuracy and efficiency in comparison with the earlier TABI-PB 1.0 code that used centroid collocation and a Taylor treecode, and in these tests, the new code ran approximately 15 times faster on a GPU than on a CPU. Finally, TABI-PB 2.0 was applied to the Zika viral capsid (1 576 628 atoms) and H1N1 viral capsid (14 442 610 atoms). The code is available on GitHub[56] and as a contributed module in the Adaptive Poisson-Boltzmann Solver (APBS) software suite.[57]

Recent studies have shown the merit of Galerkin discretization for solution of the Poisson-Boltzmann boundary integral equations,[55,58] and it would be interesting to combine that approach with the GPU-accelerated BLDTT described here. A remaining bottleneck is the NanoShaper surface triangulation, which is currently done on a CPU, and it is hoped that a GPU implementation will be forthcoming to further improve the efficiency of these calculations.

AUTHOR INFORMATION

Corresponding Authors

Leighton Wilson − *Cerebras Systems, Sunnyvale, California 94085, United States;* orcid.org/0000-0003-1676-8156; Email: leightonwilson@mac.com

Weihua Geng − *Department of Mathematics, Southern Methodist University, Dallas, Texas 75275, United States;*

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113

orcid.org/0000-0001-9911-6588; Email: wgeng@smu.edu

Robert Krasny − *Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109, United States;* orcid.org/0000-0002-8375-1699; Email: krasny@umich.edu

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jpcb.2c04604

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work was supported by National Science Foundation Grants DMS-1819094, DMS-1819193, DMS-2110767, and DMS-2110869 and Extreme Science and Engineering Discovery Environment (XSEDE) Allocation ACI-1548562. We thank Walter Rocchia for providing the H1N1 structure.

■ REFERENCES

(1) Roux, B.; Simonson, T. Implicit Solvent Models. *Biophys. Chem.* **1999**, *78*, 1−20.
(2) Zhang, Z.; Witham, S.; Alexov, E. On the Role of Electrostatics in Protein-Protein Interactions. *Phys. Biol.* **2011**, *8*, 035001.
(3) Tomasi, J. Thirty Years of Continuum Solvation Chemistry: A Review, and Prospects for the Near Future. *Theor. Chem. Acc.* **2004**, *112*, 184−203.
(4) Baker, N. A. Poisson−Boltzmann Methods for Biomolecular Electrostatics. *Methods Enzymol.* **2004**, *383*, 94−118.
(5) Lu, B.; Zhou, Y. C.; Holst, M. J.; McCammon, J. A. Recent Progress in Numerical Methods for the Poisson-Boltzmann Equation in Biophysical Applications. *Commun. Comput. Phys.* **2008**, *3*, 973−1009.
(6) Richards, F. M. Areas, Volumes, Packing, and Protein Structure. *Annu. Rev. Biophys. Bioeng.* **1977**, *6*, 151−176.
(7) Connolly, M. L. Molecular Surface Triangulation. *J. Appl. Crystallogr.* **1985**, *18*, 499−505.
(8) Warwicker, J.; Watson, H. C. Calculation of the Electric Potential in the Active Site Cleft due to α-Helix Dipoles. *J. Mol. Biol.* **1982**, *157*, 671−679.
(9) Holst, M. J.; Saied, F. Numerical solution of the Nonlinear Poisson-Boltzmann Equation: Developing More Robust and Efficient Methods. *J. Comput. Chem.* **1995**, *16*, 337−364.
(10) Baker, N. A.; Sept, D.; Joseph, S.; Holst, M. J.; McCammon, J. A. Electrostatics of Nanosystems: Application to Microtubules and the Ribosome. *Proc. Natl. Acad. Sci. U.S.A.* **2001**, *98*, 10037−10041.
(11) Luo, R.; David, L.; Gilson, M. K. Accelerated Poisson-Boltzmann Calculations for Static and Dynamic Systems. *J. Comput. Chem.* **2002**, *23*, 1244−1253.
(12) Wang, J.; Luo, R. Assessment of Linear Finite-Difference Poisson-Boltzmann Solvers. *J. Comput. Chem.* **2010**, *31*, 1689−1698.
(13) Chen, M.; Lu, B. TMSmesh: A Robust Method for Molecular Surface Mesh Generation Using a Trace Technique. *J. Chem. Theory Comput.* **2011**, *7*, 203−212.
(14) Boschitsch, A. H.; Fenley, M. O. A Fast and Robust Poisson-Boltzmann Solver Based on Adaptive Cartesian Grids. *J. Chem. Theory Comput.* **2011**, *7*, 1524−1540.
(15) Geng, W.; Wei, G. W. Multiscale Molecular Dynamics Using the Matched Interface and Boundary Method. *J. Comput. Phys.* **2011**, *230*, 435−457.
(16) Geng, W.; Zhao, S. Fully Implicit ADI Schemes for Solving the Nonlinear Poisson-Boltzmann Equation. *Mol. Based Math. Biol.* **2012**, *1*, 109−123.
(17) Geng, W.; Jacob, F. A GPU-Accelerated Direct-Sum Boundary Integral Poisson-Boltzmann Solver. *Comput. Phys. Commun.* **2013**, *184*, 1490−1496.
(18) Mirzadeh, M.; Theillard, M.; Helgadóttir, A.; Boy, D.; Gibou, F. An Adaptive, Finite Difference Solver for the Nonlinear Poisson-Boltzmann Equation with Applications to Biomolecular Computations. *Commun. Comput. Phys.* **2013**, *13*, 150−173.
(19) Wilson, L.; Zhao, S. Unconditionally Stable Time Splitting Methods for the Electrostatic Analysis of Solvated Biomolecules. *Int. J. Numer. Anal. Model.* **2016**, *13*, 852−878.
(20) Kucherova, A.; Strango, S.; Sukenik, S.; Theillard, M. Computational Modeling of Protein Conformational Changes - Application to the Opening SARS-CoV-2 Spike. *J. Comput. Phys.* **2021**, *444*, 110591.
(21) Holst, M. J.; Baker, N. A.; Wang, F. Adaptive Multilevel Finite Element Solution of the Poisson-Boltzmann Equation I: Algorithms and Examples. *J. Comput. Chem.* **2000**, *21*, 1319−1342.
(22) Baker, N. A.; Holst, M. J.; Wang, F. Adaptive Multilevel Finite Element Solution of the Poisson-Boltzmann Equation II: Refinement at Solvent-Accessible Surfaces in Biomolecular Systems. *J. Comput. Chem.* **2000**, *21*, 1343−1352.
(23) Quan, C.; Stamm, B.; Maday, Y. A Domain Decomposition Method for the Poisson-Boltzmann Solvation Models. *SIAM J. Sci. Comput.* **2019**, *41*, B320−B350.
(24) Zauhar, R. J.; Morgan, R. S. A New Method for Computing the Macromolecular Electric Potential. *J. Mol. Biol.* **1985**, *186*, 815−820.
(25) Yoon, B. J.; Lenhoff, A. M. A Boundary Element Method for Molecular Electrostatics with Electrolyte Effects. *J. Comput. Chem.* **1990**, *11*, 1080−1086.
(26) Liang, J.; Subramaniam, S. Computation of Molecular Electrostatics with Boundary Element Methods. *Biophys. J.* **1997**, *73*, 1830−1841.
(27) Juffer, A. H.; Botta, E. F. F.; van Keulen, B. A. M.; van der Ploeg, A.; Berendsen, H. J. C. The Electric Potential of a Macromolecule in a Solvent: A Fundamental Approach. *J. Comput. Phys.* **1991**, *97*, 144−171.
(28) Boschitsch, A. H.; Fenley, M. O.; Zhou, H.-X. Fast Boundary Element Method for the Linear Poisson-Boltzmann Equation. *J. Phys. Chem. B* **2002**, *106*, 2741−2754.
(29) Lu, B.; Cheng, X.; Huang, J.; McCammon, J. A. Order *N* Algorithm for Computation of Electrostatic Interactions in Biomolecular Systems. *Proc. Natl. Acad. Sci. U.S.A.* **2006**, *103*, 19314−19319.
(30) Lu, B.; McCammon, J. A. Improved Boundary Element Methods for Poisson-Boltzmann Electrostatic Potential and Force Calculations. *J. Chem. Theory Comput.* **2007**, *3*, 1134−1142.
(31) Geng, W.; Krasny, R. A Treecode-Accelerated Boundary Integral Poisson-Boltzmann Solver for Electrostatics of Solvated Biomolecules. *J. Comput. Phys.* **2013**, *247*, 62−78.
(32) Cooper, C. D.; Bardhan, J. P.; Barba, L. A. A Biomolecular Electrostatics Solver Using Python, GPUs and Boundary Elements That Can Handle Solvent-Filled Cavities and Stern Layers. *Comput. Phys. Commun.* **2014**, *185*, 720−729.
(33) Zhong, Y.; Ren, K.; Tsai, R. An Implicit Boundary Integral Method for Computing Electric Potential of Macromolecules in Solvent. *J. Comput. Phys.* **2018**, *359*, 199−215.
(34) Geng, W. A. Boundary Integral Poisson-Boltzmann Solvers Package for Solvated Bimolecular Simulations. *Mol. Based Math. Biol.* **2015**, *3*, 54−69.
(35) Sanner, M. F.; Olson, A. J.; Spehner, J.-C. Fast and Robust Computation of Molecular Surfaces. *Proc. 11th ACM Symp. Comput. Geom. (SoCG)* **1995**, C6−C7.
(36) Sanner, M. F.; Olson, A. J.; Spehner, J.-C. Reduced Surface: An Efficient Way to Compute Molecular Surfaces. *Biopolymers* **1996**, *38*, 305−320.
(37) Saad, Y.; Schultz, M. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.* **1986**, *7*, 856−869.
(38) Li, P.; Johnston, H.; Krasny, R. A Cartesian Treecode for Screened Coulomb Interactions. *J. Comput. Phys.* **2009**, *228*, 3858−3868.

(39) Decherchi, S.; Rocchia, W. A General and Robust Ray-Casting-Based Algorithm for Triangulating Surfaces at the Nanoscale. *PLoS One* **2013**, *8*, No. e59744.
(40) Decherchi, S.; Colmenares, J.; Catalano, C. E.; Spagnuolo, M.; Alexov, E.; Rocchia, W. Between Algorithm and Model: Different Molecular Surface Definitions for the Poisson-Boltzmann Based Electrostatic Characterization of Biomolecules in Solution. *Commun. Comput. Phys.* **2013**, *13*, 61−89.
(41) Decherchi, S.; Spitaleri, A.; Stone, J.; Rocchia, W. NanoShaper-VMD Interface: Computing and Visualizing Surfaces, Pockets and Channels in Molecular Systems. *Bioinformatics* **2019**, *35*, 1241−1243.
(42) Chen, J.; Geng, W. On Preconditioning the Treecode-Accelerated Boundary Integral (TABI) Poisson-Boltzmann Solver. *J. Comput. Phys.* **2018**, *373*, 750−762.
(43) Wilson, L.; Vaughn, N.; Krasny, R. A GPU-Accelerated Fast Summation Method Based on Barycentric Lagrange Interpolation and Dual Tree Traversal. *Comput. Phys. Commun.* **2021**, *265*, 108017.
(44) Wilson, L.; Krasny, R. Comparison of the MSMS and NanoShaper Molecular Surface Triangulation Codes in the TABI Poisson-Boltzmann Solver. *J. Comput. Chem.* **2021**, *42*, 1552−1560.
(45) Boateng, H. A. *Cartesian Treecode Algorithms for Electrostatic Interactions in Molecular Dynamics Simulations*. Ph.D. thesis, University of Michigan: Ann Arbor, MI, 2010.
(46) Boateng, H. A.; Krasny, R. Comparison of Treecodes for Computing Electrostatic Potentials in Charged Particle Systems with Disjoint Targets and Sources. *J. Comput. Chem.* **2013**, *34*, 2159−2167.
(47) Berrut, J.-P.; Trefethen, L. N. Barycentric Lagrange Interpolation. *SIAM Rev.* **2004**, *46*, 501−517.
(48) Greengard, L.; Rokhlin, V. A Fast Algorithm for Particle Simulations. *J. Comput. Phys.* **1987**, *73*, 325−348.
(49) Appel, A. W. An Efficient Program for Many-Body Simulation. *SIAM J. Sci. Stat. Comput.* **1985**, *6*, 85−103.
(50) Dehnen, W. A. Hierarchical *O*(*N*) Force Calculation Algorithm. *J. Comput. Phys.* **2002**, *179*, 27−42.
(51) Vaughn, N.; Wilson, L.; Krasny, R. *A GPU-Accelerated Barycentric Lagrange Treecode. 2020 IEEE International Parallel and Distributed Processing Symposium Workshops*; IPDPSW: 2020; pp 701−710.
(52) Dolinsky, T. J.; Nielsen, J. E.; McCammon, J. A.; Baker, N. A. PDB2PQR: An Automated Pipeline for the Setup, Execution, and Analysis of Poisson-Boltzmann Electrostatics Calculations. *Nucleic Acids Res.* **2004**, *32*, W665−W667.
(53) Sevvana, M.; Long, F.; Miller, A. S.; Klose, T.; Buda, G.; Sun, L.; Kuhn, R. J.; Rossmann, M. G. Refinement and Analysis of the Mature Zika Virus Cryo-EM Structure at 3.1 Å Resolution. *Structure* **2018**, *26*, 1169−1177.
(54) Amaro, R. E.; Ieong, P. U.; Huber, G.; Dommer, A.; Steven, A. C.; Bush, R. M.; Durrant, J. D.; Votapka, L. W. A Computational Assay that Explores the Hemagglutinin/Neuraminidase Functional Balance Reveals the Neuraminidase Secondary Site as a Novel Anti-Influenza Target. *ACS Cent. Sci.* **2018**, *4*, 1570−1577.
(55) Wang, T.; Cooper, C. D.; Betcke, T.; Barba, L. A. *High-Productivity, High-Performance Workflow for Virus-Scale Electrostatic Simulations with Bempp-Exafmm*. 2021; arXiv:2103.01048v2.
(56) Wilson, L. W.; Krasny, R.; Geng, W.; Chen, J. *TABI-PB 2.0*; http://github.com/Treecodes/TABI-PB (accessed August 27, 2022).
(57) Jurrus, E.; Engel, D.; Star, K.; Monson, K.; Brandi, J.; Felberg, L. E.; Brookes, D. H.; Wilson, L.; Chen, J.; Liles, K.; et al. Improvements to the APBS Biomolecular Solvation Software Suite. *Protein Sci.* **2018**, *27*, 112−128.
(58) Chen, J.; Tausch, J.; Geng, W. *A Cartesian FMM-Accelerated Galerkin Boundary Integral Poisson-Boltzmann Solver*. 2021; arXiv:2110.13778.

https://doi.org/10.1021/acs.jpcb.2c04604
J. Phys. Chem. B 2022, 126, 7104−7113