Computer Physics Communications 184 (2013) 1490–1496

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

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

A GPU-accelerated direct-sum boundary integral Poisson–Boltzmann solver

Weihua Geng [a,*], Ferosh Jacob [b]

[a] Department of Mathematics, University of Alabama, Tuscaloosa, AL 35487, USA
[b] Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487, USA

ARTICLE INFO

Article history:
Received 9 December 2012
Received in revised form
19 January 2013
Accepted 23 January 2013
Available online 1 February 2013

Keywords:
Poisson–Boltzmann
Electrostatics
Boundary integral
Parallel computing
Graphic processing units (GPU)

ABSTRACT

In this paper, we present a GPU-accelerated direct-sum boundary integral method to solve the linear Poisson–Boltzmann (PB) equation. In our method, a well-posed boundary integral formulation is used to ensure the fast convergence of Krylov subspace based linear algebraic solver such as the GMRES. The molecular surfaces are discretized with flat triangles and centroid collocation. To speed up our method, we take advantage of the parallel nature of the boundary integral formulation and parallelize the schemes within CUDA shared memory architecture on GPU. The schemes use only $11N+6N_c$ size-of-double device memory for a biomolecule with N triangular surface elements and N_c partial charges. Numerical tests of these schemes show well-maintained accuracy and fast convergence. The GPU implementation using one GPU card (Nvidia Tesla M2070) achieves 120–150X speed-up to the implementation using one CPU (Intel L5640 2.27 GHz). With our approach, solving PB equations on well-discretized molecular surfaces with up to 300,000 boundary elements will take less than about 10 min, hence our approach is particularly suitable for fast electrostatics computations on small to medium biomolecules.

1. Introduction

Molecular mechanics uses Newton's classical mechanics to model molecular systems. The potential energy of all systems in molecular mechanics is calculated using force fields. Among all components of the force fields, electrostatics are critical due to their ubiquitous existence and are expensive to compute since they are long-range pairwise interactions. Poisson–Boltzmann (PB) model is an effective alternative for resolving electrostatics that includes energy, potential and forces of solvated biomolecules [1]. As an implicit solvent approach, the PB model uses a mean field approximation to trace the solvent effects and applies Boltzmann distribution to model the mobile ions. These implicit treatments make the PB model computationally more efficient compared to explicit solvent models, in which atomic details of solvent molecules and mobile ions are explicitly described.

In the PB model, the computational domain $\mathbb{R}^3$ is divided into the solute domain Ω_1 and the solvent domain Ω_2 by a closed molecular surface Γ such that $\mathbb{R}^3 = \Omega_1 \cup \Omega_2 \cup \Gamma$. The molecular surface Γ is formed by the traces of a spherical solvent probe rolling in contact with the van del Walls balls of the solute atoms [2,3]. The molecule, which is located in domain Ω_1 with dielectric constant ε_1, is represented by a set of N_c point charges carrying Q_i charge in the units of e_c, the elementary charge, at positions $\mathbf{x}_i$, $i = 1, \ldots, N_c$. The exterior domain contains the solvent with dielectric constant ε_2, as well as mobile ions. For $\mathbf{x} = (x, y, z)$, the PB equation for the electrostatic potential in each domain is derived from Gauss's law and the Boltzmann distribution. Assuming weak ionic strength (e.g., the concentration of the physiological saline in a room temperature), the linearized PB equation and its interface jump conditions and boundary conditions have the forms

$$\nabla \cdot (\varepsilon_1(\mathbf{x})\nabla\phi_1(\mathbf{x})) = -\sum_{i=1}^{N_c} q_i \delta(\mathbf{x}-\mathbf{x}_i) \quad \text{in } \Omega_1, \tag{1}$$

$$\nabla \cdot (\varepsilon_2(\mathbf{x})\nabla\phi_2(\mathbf{x})) - \kappa^2\phi_2(\mathbf{x}) = 0 \quad \text{in } \Omega_2, \tag{2}$$

$$\phi_1(\mathbf{x}) = \phi_2(\mathbf{x}),$$
$$\varepsilon_1 \frac{\partial\phi_1(\mathbf{x})}{\partial\nu} = \varepsilon_2 \frac{\partial\phi_2(\mathbf{x})}{\partial\nu} \quad \text{on } \Gamma = \partial\Omega_1 = \partial\Omega_2, \tag{3}$$

$$\lim_{|\mathbf{x}|\to\infty} \phi_2(\mathbf{x}) = 0, \tag{4}$$

where ϕ_1 and ϕ_2 are the electrostatic potentials in each domain, $q_i = e_c Q_i/k_B T$, $i = 1, \ldots, N_c$, e_c the electron charge, k_B the Boltzmann's constant, T the absolute temperature, δ the Dirac delta function, κ the Debye–Hückel parameter, and ν the unit outward normal on the interface Γ. Note the $\kappa^2\phi(\mathbf{x})$ term in Eq. (2) is the linearized form of $\kappa^2 \sinh(\phi(\mathbf{x}))$ when weak ionic strength is assumed.

* Corresponding author. Tel.: +1 205 3485302; fax: +1 205 3487067.
E-mail address: wgeng@as.ua.edu (W. Geng).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2013.01.017

The PB equation is an elliptic equation defined on multiple domains with discontinuous coefficients across the domain interface. The PB equation has analytical solutions only for the simple geometries such as spheres [4] or rods [5]. For molecules with complex geometries, the PB equation can only be solved numerically, which is challenging due to the non-smoothness of the solution subject to Eq. (3), the complex geometry of the interface Γ, the singular partial charges in Eq. (1), and the boundary conditions at the infinity in Eq. (4). Many numerical PB solvers were developed and they can be roughly divided into two categories: (1) the 3-D mesh-based finite difference/finite element methods [6–11]; and (2) the boundary integral methods [12–21]. All these methods have their own advantages and disadvantages. For example, the PB solvers embedded in molecular modeling packages, such as Delphi [6], CHARMM [7], AMBER [8], and APBS [9] use standard seven-point finite difference to discretize the PB equation. Although standard finite difference methods arguably result in reduced accuracy due to the smoothened treatment of Eq. (3), the efficient, robust and user-friendly features of these PB solvers brought their popularity to the computational biophysics/biochemistry community. Compared with 3-D mesh-based methods, the boundary integral methods have advantages since they impose the singular partial charge in Eq. (1) and the far-field boundary condition in Eq. (4) exactly, use appropriate boundary elements to represent surface geometry to desired precision, and enforce the continuity condition in Eq. (3) across the interface explicitly. They are naturally more convenient methods to attack numerical difficulties in solving the PB equation.

An often claimed advantage of boundary integral methods compared to finite difference methods is the reduction of the 3-D differential equation to a 2-D surface integral equation. However, the finite difference methods generate a 7-band sparse matrix, while the boundary integral methods constitute a fully dense matrix, which is prohibitively expensive to store and whose matrix–vector product is computationally costly. The remedy is to generate the matrix on-the-fly and compute the matrix–vector product with the assistance of fast algorithms for N-body problems, such as Fast Multipole Methods (FMM) [13–15,21] and treecode [17]. In the last few decades, the advent of the multicore computers and related parallel architectures such as MPI and Open MP brought the boundary integral methods to a superior position to 3-D mesh-based methods. Recently, the appearance of GPU computing further boosts the boundary element methods, particularly for schemes that have simpler algorithms and use less memory.

GPU computing refers to the use of graphics processing units for scientific and engineering computing applications rather than traditionally rendering graphs. A GPU card can be treated as an array of many simplified CPUs with reduced but concurrent computing power and more limited but faster memory access to CPUs. GPUs execute many concurrent threads relatively slowly, rather than a single thread quickly. This means that GPU computing is more suitable for problems with high concurrency, straightforward workflow, low memory requirements, and infrequent communication. GPUs have broad areas of application, particularly in speeding up molecular modeling. Molecular dynamics packages such as AMBER [22,23] and NAMD [24,25] have GPU implementations that achieve significant speedup compared to CPU implementations.

The matrix–vector product computed in boundary integral formulation is similar to computing N-body problems for particle interactions. Solving N-body problems and related applications in boundary integral methods are popular targets for GPU computing. For examples, Nyland et al. [26] computed N-body interactions by direct summation, Burtscher et al. [27] used the Barnes–Hut treecode [28] to compute the dynamics of 5 million point masses on a 1.3 GHz GPU with 240 threads, obtaining a speedup of 74 over a 2.53 GHz CPU, and Yokota et al. [29] demonstrated a FMM accelerated boundary integral PB solver on 512 GPUs, achieving 34.6 TFlops.

In this paper, we present a parallel boundary integral PB solver on GPUs. We adopt the well-posed formulation from Juffer et al. [12], rather than the straightforward integral formulation presented in [30] and applied in [16,29]. The N-body summation is computed directly, therefore the scheme is implementation convenient and memory saving for GPU computing. The rest of the paper is organized as follows. In Section 2, we provide our algorithms including the well-posed boundary integral formulation and its discretization, followed by CUDA implementation on GPUs. In Section 3, we present the numerical results, first on the spherical cavities with multiple partial changes, whose analytical solutions are available and then on a series of proteins with various sizes and geometries. This paper ends with a section of concluding remarks.

2. Methods

We use the well-posed boundary integral formulation from Juffer's work [12] together with a flat triangulation and a centroid collocation. A factor affecting the accuracy of boundary integral method is the discretization of the surface. We use a non-uniformed triangular surface from MSMS [31]. Throughout this paper, we call our GPU-accelerated boundary integral Poisson–Boltzmann solver as GABI-PB solver.

2.1. Well-posed integral formulation

The differential PB equations (1) and (2) can be converted to boundary integral equations. In order to do this, we first define the fundamental solutions to the Poisson equation (1) in Ω_1 and the fundamental solution to the PB equation (2) in Ω_2 as

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

Note $G_0(\mathbf{x},\mathbf{y})$ and $G_\kappa(\mathbf{x},\mathbf{y})$ are called Coulomb and screened Coulomb potentials in electrostatic theory. By applying Green's second theorem, and canceling the normal derivative terms with interface conditions in Eq. (3), the coupled integral equations can be derived as [30]:

$$\begin{aligned}\phi_1(\mathbf{x}) &= \int_\Gamma \left[G_0(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu_\mathbf{y}} - \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi_1(\mathbf{y}) \right] dS_\mathbf{y} \\ &\quad + \sum_{k=1}^{N_c} q_k G_0(\mathbf{x},\mathbf{y}_k), \quad \mathbf{x} \in \Omega_1, \end{aligned} \tag{6}$$

$$\begin{aligned}\phi_2(\mathbf{x}) &= \int_\Gamma \left[-G_\kappa(\mathbf{x},\mathbf{y}) \frac{\partial \phi_2(\mathbf{y})}{\partial \nu_\mathbf{y}} + \frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial \nu_\mathbf{y}} \phi_2(\mathbf{y}) \right] dS_\mathbf{y}, \\ &\quad \mathbf{x} \in \Omega_2. \end{aligned} \tag{7}$$

However, straightforward discretization of Eqs. (6) and (7) yields an ill-conditioned linear system, whose condition number dramatically increases as the number of boundary elements increases [32]. Juffer et al. derived a well-posed boundary integral formulation by going through the differentiation of the single-layer potentials and the double-layer potentials [12]. Here the single-layer potentials are from the induced point charge distributions G_0 and G_κ on surface Γ, while the double-layer potential are from the induced dipole charge distributions, which are the normal derivatives of G_0 and G_κ on surface Γ. The desired integral forms are as:

$$\begin{aligned}&\frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}) \\ &= \int_\Gamma \left[K_1(\mathbf{x},\mathbf{y}) \frac{\partial \phi_1(\mathbf{y})}{\partial \nu_\mathbf{y}} + K_2(\mathbf{x},\mathbf{y}) \phi_1(\mathbf{y}) \right] dS_\mathbf{y}\end{aligned}$$

$$+S_1(\mathbf{x}), \quad \mathbf{x}\in\Gamma, \tag{8}$$

$$\frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial\phi_1(\mathbf{x})}{\partial\nu_{\mathbf{x}}} = \int_\Gamma\left[K_3(\mathbf{x},\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu_{\mathbf{y}}}+K_4(\mathbf{x},\mathbf{y})\phi_1(\mathbf{y})\right]dS_{\mathbf{y}} +S_2(\mathbf{x}), \quad \mathbf{x}\in\Gamma, \tag{9}$$

with the notation for the kernels and the source terms.

$$\begin{aligned} K_1(\mathbf{x},\mathbf{y}) &= G_0(\mathbf{x},\mathbf{y})-G_\kappa(\mathbf{x},\mathbf{y}),\\ K_2(\mathbf{x},\mathbf{y}) &= \varepsilon\frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{y}}}-\frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{y}}},\\ K_3(\mathbf{x},\mathbf{y}) &= \frac{\partial G_0(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{x}}}-\frac{1}{\varepsilon}\frac{\partial G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{x}}},\\ K_4(\mathbf{x},\mathbf{y}) &= \frac{\partial^2 G_\kappa(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{x}}\partial\nu_{\mathbf{y}}}-\frac{\partial^2 G_0(\mathbf{x},\mathbf{y})}{\partial\nu_{\mathbf{x}}\partial\nu_{\mathbf{y}}}, \end{aligned} \tag{10}$$

$$S_1(\mathbf{x})=\sum_{k=1}^{N_c}q_kG_0(\mathbf{x},\mathbf{y}_k), \qquad S_2(\mathbf{x})=\sum_{k=1}^{N_c}q_k\frac{\partial G_0(\mathbf{x},\mathbf{y}_k)}{\partial\nu_{\mathbf{x}}} \tag{11}$$

and $\varepsilon=\varepsilon_1/\varepsilon_2$. Note, this is the well-posed Fredholm second kind of integral equation, which is also our choice in this paper.

2.2. Discretization

We discretize the integral Eqs. (8) and (9) with the flat triangle and the centroid collocation (the quadrature point is located at the center of each triangle). This scheme also assumes that the potential and its normal derivative, as well as the kernel functions are uniform on each triangle. When the singularities in kernels occur ($\mathbf{x}=\mathbf{y}$), the contribution of this triangle in the integral is then simply removed. This scheme, which provides the convenience of incorporating fast algorithms, such as FMM [32] and treecode [17], is in fact widely used in the latest boundary integral methods in solving PB equations.

In this paper, suppose the triangulation program MSMS [31] discretizes the molecular surface to a set of N triangular elements connected with N_v vertices and N_e edges. We then have the relation $N_v+N-N_e=2$ called Euler's polyhedron formula. With potential $\phi_1(\mathbf{x})$ and its normal derivative $\frac{\partial\phi_1(\mathbf{x})}{\partial\nu_{\mathbf{x}}}$ at the centroid of each triangular element as the unknowns, Eqs. (8) and (9) are converted to a linear algebraic system $\mathbf{Au}=\mathbf{b}$, whose elements are specified as follows. Note, we do not explicitly express $\mathbf{A}$ in an iterative method as we only concern $\mathbf{Au}$ on each iteration.

For $i=1,2,\ldots,N$, the ith and the $(i+N)$th element of the discretized matrix–vector product $\mathbf{Au}$ are given as

$$\{\mathbf{Au}\}_i = \frac{1}{2}(1+\varepsilon)\phi_1(\mathbf{x}_i) - \sum_{j=1,j\neq i}^{N}W_j\left[K_1(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu_{\mathbf{x}_j}}+K_2(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right] \tag{12}$$

$$\{\mathbf{Au}\}_{i+N} = \frac{1}{2}\left(1+\frac{1}{\varepsilon}\right)\frac{\partial\phi_1(\mathbf{x}_i)}{\partial\nu_{\mathbf{x}_i}} - \sum_{j=1,j\neq i}^{N}W_j\left[K_3(\mathbf{x}_i,\mathbf{x}_j)\frac{\partial\phi_1(\mathbf{x}_j)}{\partial\nu_{\mathbf{x}_j}}+K_4(\mathbf{x}_i,\mathbf{x}_j)\phi_1(\mathbf{x}_j)\right] \tag{13}$$

where W_j is the area of the jth element. Note, we use $\mathbf{x}_i$ and $\mathbf{x}_j$ (instead of $\mathbf{y}_j$) in all kernels to indicate sources and targets are the same set of points. The expressions of $\mathbf{b}_i$ and $\mathbf{b}_{i+N}$ are directly obtained from S_1 and S_2 in Eq. (11). In producing the results of this paper, we apply GMRES solver [33] to solve the linear algebraic system from the discretization of Eqs. (8) and (9), which requires computing the matrix–vector product in Eqs. (12) and (13) on each iteration.

Table 1
Pseudocode for GABI-PB solver using GPU.

```
1   On host (CPU)
2       read biomolecule data (charge and structure)
3       call MSMS to generate triangulation
4       copy biomolecule data and triangulation to device
5   On device (GPU)
6       each thread concurrently computes and stores source terms for
        assigned triangles
7       copy source terms on device to host
8   On host
9       set initial guess x_0 for GMRES iteration and copy it to device
10  On device
11      each thread concurrently computes assigned segment of
        matrix–vector product y = Ax
12      copy the computed matrix–vector y to host memory
13  On host
14      test for GMRES convergence
15          if no, generate new x and copy it to device, go to step 10 for the
            next iteration
16          if yes, generate and copy the final solution to device and go to
            step 17
17  On device
18      compute assigned segment of electrostatic solvation energy
19      copy results in step 18 to host
20  On host
21      add segments of electrostatic solvation energy and output result
```

2.3. Electrostatic solvation energy formulation

To perform the solvation analysis of interested biomolecules, the electrostatic solvation energy is computed by

$$E_{\text{sol}} = \frac{1}{2}\sum_{k=1}^{N_c}q_k\phi_{\text{reac}}(\mathbf{x}_k) = \frac{1}{2}\sum_{k=1}^{N_c}q_k\int_\Gamma\left[K_1(\mathbf{x}_k,\mathbf{y})\frac{\partial\phi_1(\mathbf{y})}{\partial\nu_{\mathbf{y}}}+K_2(\mathbf{x}_k,\mathbf{y})\phi_1(\mathbf{y})\right]dS_{\mathbf{y}}, \tag{14}$$

where $\phi_{\text{reac}}(\mathbf{x}_k)=\phi_1(\mathbf{x}_k)-S_1(\mathbf{x}_k)$, whose formulation is the integral part of Eq. (14), is the reaction potential at the kth solute atom. The electrostatic solvation energy, which can be regarded as the atomistic charge q_k weighted average of the reaction potential ϕ_{reac}, can effectively characterize the accuracy of a PB solver.

2.4. GPU/CUDA implementation

The GABI-PB solver uses the boundary integral formulation, which can be conveniently parallelized. The majority of the computing time is taken by the following subroutines.

(1) Compute the source term in Eqs. (8) and (9).
(2) Perform matrix–vector product in Eqs. (12) and (13) on each GMRES iteration.
(3) Compute electrostatic solvation energy.

Among all of these subroutines, subroutine (2) is the most expensive one and is repeatedly computed on each GMRES iteration. We compute all of these routines in parallel, to minimize the computation time. Table 1 provides the pseudo code for the overall computation.

In this pseudo code, we divide all the operations to those on host performed by the CPU and those on device performed by the GPU. We use N_c to denote the number of atoms of the biomolecule and N to represent the number of triangular elements. In the following description, we include the memory use in a pair of parentheses following each variable we would have claimed.

We first read in the atomistic coordinates ($3N_c$), radii (N_c) and charges (N_c) of the given biomolecule (step 2) and call the MSMS [31] program to generate triangulation (step 3) including the faces and the vertices of the triangulation. We convert these

triangulation information to centroid coordinates ($3N$), normal direction vectors ($3N$) and triangle areas (N), and copy them together with the atomistic coordinates, radii and charges to the device (step 4). So far, the device memory use is $5N_c + 7N$ size-of-double. We then use multi-threads on device to compute the source terms ($2N$, but deallocated right after being copied to host) in Eqs. (8) and (9) (step 6) and copy them to the host (step 7). Next, we set the initial guess $\mathbf{x}_0$ of the solution (currently a zero vector) and copy it to device (step 9). Note that, when the GMRES is restarted (usually this needs to be done every 10–20 steps to ensure the small number of terms in Krylov subspace expansion), we use the approximated solution achieved from the previous iteration as the initial guess $\mathbf{x}_0$.

What follows is the major computing part. We compute the matrix–vector product $\mathbf{y} = \mathbf{Ax}$ (step 11) on multi-threads GPU device, and copy the resulting $\mathbf{y}$ to the host (step 12). These two steps take additionally $4N$ size-of-double device memory. Then on host, we test if the obtained $\mathbf{y}$ satisfies the GMRES convergence criterion (step 14) to decide whether to start the next iteration (step 15) or terminate the GMRES and generate the final solution (step 16). After GMRES convergence criterion is satisfied, we need to compute the atom-wise components of the electrostatic solvation energy on device, using the final solution copied from the host (step 18). This step takes additional N_c size-of-double device memory. Finally, with the components of the electrostatic solvation energy copied from device (step 19), we compute the total electrostatic solvation energy on host (step 21). All together, we have allocated $6N_c + 11N$ size-of-double device memory.

In implementing algorithms, we tune the CUDA code with some optimization strategies like loop unrolling, coalesced memory access (using double3 data type instead of three double variables for some structures like position and charge), and fast CUDA operators like "rsqrt". We also test and choose the optimized value 256 as the number of thread per block. Although these strategies did not substantially improve the performance, they contribute to the overall performance.

3. Results

In this section, we present the numerical results. We first solve the PB equation on a spherical cavity with multiple charges at different locations. The analytical solutions in terms of spherical harmonics are available [4]. We then solve the PB equation on a series of 24 proteins with different sizes and geometries. These protein structures are downloaded from protein data bank (PDB), and their charges and hydrogens are added with CHARMM22 force field [34]. We report the electrostatic solvation energy results and its associated execution time for solving the PB equation and computing electrostatic solvation energy on these proteins with and without GPU acceleration to demonstrate the improved efficiency from using GPU. All algorithms are written in C and CUDA and compiled by gcc with flag "-O3" and nvcc with flag "-O3 -arch = sm_20 -use_fast_math". The simulations are performed with a single CPU (Intel(R) Xeon(R) CPU E5640 @ 2.27 GHz with 2G Memory) on a 12-core workstation and one GPU (Nvidia Tesla M2070) card. Before we reveal the numerical results, we define order and error.

3.1. Order and error

In this paper, we report the relative L_∞ error of the surface potential, which is defined as

$$e_\phi = \frac{\max\limits_{i=1,\ldots,N} |\phi^{\text{num}}(x_i) - \phi^{\text{exa}}(x_i)|}{\max\limits_{i=1,\ldots,N} |\phi^{\text{exa}}(x_i)|} \tag{15}$$

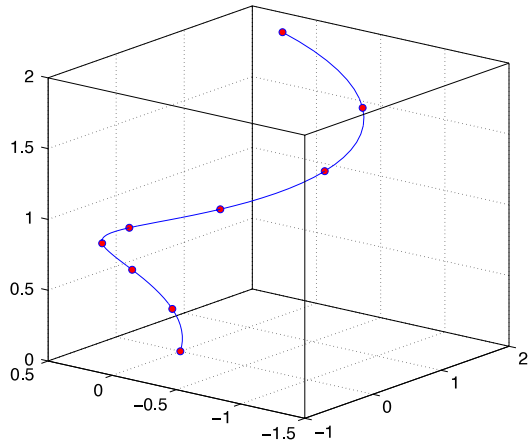

Fig. 1. Locations of the partial charges inside a spherical cavity with radius $r = 4$ Å.

where N is the number of unknowns, and also the number of triangular elements of a particular discretized molecular surface. The notation ϕ^{num} represents numerically solved surface potential and ϕ^{exa} denotes the analytical solutions obtained by Kirkwood's spherical harmonic expansion [4]. The discretization on the molecular surface has a parameter "density", number of vertices per Å^2.

The numerical order of accuracy is computed with

$$\texttt{order} = \log_{\frac{\texttt{coarse\_mesh}}{\texttt{fine\_mesh}}} \frac{e_\phi^{\texttt{coarse}}}{e_\phi^{\texttt{fine}}} \tag{16}$$

following the convention of numerical analysis, where "mesh" refers to density for boundary integral methods at both coarse and fine levels.

3.2. Accuracy tests on a spherical cavity

We first perform the numerical tests for the accuracy of the algorithm on a spherical cavity, where the analytical solution in terms of spherical harmonics expansion is available [4]. The test case we designed is a sphere of radius $r = 4$Å containing nine partial charges, which are located along the space curve $\mathbf{r}(t) = \langle \frac{3}{4\pi}\cos t, \frac{3}{4\pi}\sin t, \frac{t}{\pi}\rangle$ for $t = 0, \frac{\pi}{2}, \ldots, 2\pi$. These nine point charges carry 0.1, 0.2, ..., 0.9 electric charges respectively in the units of e_c, the elementary charge. We plot the charge locations in Fig. 1, which resemble a segment of a biological helix. Table 2 reports the numerical results. In the first column of the table, we increase the MSMS input parameter "density" (number of vertices per Å^2) by doubling its current value each time. The number of triangular elements are also approximately doubled each time as seen in column 2. The electrostatic solvation energy are reported in column 3 and we can see a consistent pattern that these values are approaching the true value −952.52 kcal/mol. Column 4 is the relative L_∞ error of the surface potential, whose convergence pattern can be better seen from column 5 in terms of orders. Note that the 0.5th order observed here is relative to the *area* of the triangular element. If the 1-D *length* is considered, the order should be about one.

3.3. Accuracy and efficiency test on proteins

We next solve the PB equation and compute the electrostatic solvation energy on a series of proteins with different sizes and geometries. The numerical results are reported in Table 3. In this table, the first column is the index for the convenience of identification. The second row is the four-digit protein data bank (PDB) ID of the corresponding protein. Column 3 is the number

Table 2
Accuracy tests on a spherical cavity: radius $r = 4$ Å, nine charges are located on the space curve $\langle \frac{3}{4\pi}\cos t, \frac{3}{4\pi}\sin t, \frac{t}{\pi}\rangle$ for $t = 0 : \frac{\pi}{2} : 2\pi$; e_ϕ is the relative L_∞ error in surface potential ϕ; order is relative to the area of the elements; the exact value of E_{sol} is −952.52 kcal/mol.

Density	#of ele.	E_{sol}	e_ϕ	Order
1	370	−971.92	4.20E−02	–
2	736	−968.05	2.80E−02	0.58
4	1 572	−964.03	1.85E−02	0.60
8	3 124	−961.08	1.28E−02	0.53
16	6 308	−958.75	8.90E−03	0.52
32	12 772	−956.99	6.24E−03	0.51
64	25 494	−955.74	4.35E−03	0.52
128	51 204	−954.81	3.06E−03	0.51

of triangular elements when the molecular surfaces of these proteins are discretized. We uniformly choose "density = 10" so that proteins with larger molecular surface areas as seen in column 5 will normally generate larger number of elements. A few exceptions happen when MSMS modifies the given density to fit its triangulation needs, resulting in a slightly mismatched order for the data in columns 3 and 5. Column 4 gives the number of atoms of associated proteins. We can see in most of the cases the larger number of atoms results in the larger molecular surface area and number of elements. Column 6 lists the number of iterations. From this column, we noticed the GABI-PB solver converges within 20 steps on the majority of the proteins and the number of iterations does not increase with the increment of the number of elements, which contributes to the well-posed integral formulation. Occasionally, the number of iteration is high, for example for the 7th, 8th, and 24th proteins. This largely attributes the fact that the MSMS triangulations of these proteins have triangles with very small areas or very short sides. Currently we are working on a project to improve the triangulation. We also plot the number of iterations vs the number of elements in Fig. 2(a) on which the fact that most numbers of iterations are consistently low with a few exceptions is visualized. Column 7 reports the solvation energies. We compare these values with results from our previous work in [35] and consistency of results are observed. Column 8 shows the time for solving PB equation and computing the electrostatic solvation energy on one CPU (T_1 in seconds), Column 9 displays the time for that with GPU acceleration (T_p in seconds), and Column 10 reports the ratio between them. From column 9, we can see most of the jobs are finished within 2–3 min except jobs on proteins with slow convergence (e.g. the 7th, 8th and 24th proteins). The ratio in the last column demonstrates the overall 120–150X parallel speedup.

We also plot the results of the last three columns in Fig. 2. From Fig. 2(b), we see the CPU time T_1 is consistently a scale of 100+ of the GPU time T_p. Time increases as the number of elements increases normally. Fig. 2(c) enables us to better observe that the speed-up increases as the number of elements increases, which is advantageous for solving problems of larger sizes.

Next we focus on two proteins: 1frd and 1svr. Results of these two proteins in Table 3 show fast convergence with number of iterations around 10 and they have surface areas more or less of 4500 Å^2 and number of elements is about 80,000 at "density = 10". To test the accuracy and efficiency of GABI-PB solver on these two proteins, we solve the PB equation and compute the electrostatic solvation energy at different "densities" ranging from 1 to 32 by doubling its value each time. The numerical results are shown in Table 4. We can see that as the densities are doubled each time, the number of elements are doubled approximately. The solvation energies at different densities are approaching to the values at the largest density 32. The running time T_p with GPU acceleration increases at the rate of $\mathcal{O}(N^2)$ after the number of elements are sufficiently large. Both cases show that, for solving PB equation and computing electrostatic solvation energy on molecular surfaces discretized with nearly 300,000 elements, the time required is only about 10 min. The number of iteration in Table 4 shows that the finer resolution for the same molecular surface will not increase the number of iterations. To further investigate the convergence of accuracy on proteins, we plot the electrostatic solvation energy on Fig. 3. By using cubic interpolation, we can see the electrostatic solvation energy computed for both proteins eventually converge toward its interpolated value (the red "∗").

4. Conclusion

This paper describes a direct summation based GPU-accelerated boundary integral Poisson–Boltzmann (GABI-PB) solver. This solver discretizes the molecular surfaces with flat triangles and

Table 3
Numerical results for computing the electrostatic solvation energy of 24 proteins: N is the number of elements, N_c is the number of atoms, N_{it} is the number of iterations, area is the molecular surface area, E_{sol} is the electrostatic solvation energy, T_1 is the running time on one CPU, and T_p is the running time with GPU acceleration.

ID	PDB	N	N_c	Area (Å^2)	N_{it}	E_{sol} (kcal/mol)	CPU T_1 (s)	GPU T_p (s)	T_1/T_p
1	1ajj	40 496	519	2176	8	−1145.76	1 323	11	125
2	2erl	43 214	573	2329	8	−961.68	1 410	11	124
3	1cbn	44 367	648	2377	8	−307.03	1 486	12	129
4	1vii	47 070	596	2488	11	−915.18	2 501	19	129
5	1fca	47 461	729	2558	8	−1215.34	1 699	13	127
6	1bbl	49 071	576	2599	10	−1001.47	2 267	17	134
7	2pde	50 518	667	2727	99	−824.91	25 985	194	134
8	1sh1	51 186	702	2756	100	−760.71	27 160	201	135
9	1vjw	52 536	828	2799	8	−1255.93	2 085	16	133
10	1uxc	53 602	809	2848	9	−1154.60	2 437	18	136
11	1ptq	54 260	795	2910	10	−883.77	2 778	21	130
12	1bor	54 629	832	2910	12	−863.91	3 657	28	132
13	1fxd	54 692	824	2935	7	−3344.56	1 963	15	129
14	1r69	57 646	997	3068	9	−1100.83	2 823	22	131
15	1mbg	58 473	903	3085	9	−1368.58	2 901	22	132
16	1bpi	60 600	898	3247	24	−1320.89	9 004	64	141
17	1hpt	61 164	858	3277	11	−825.59	4 248	32	133
18	451c	79 202	1216	4778	19	−1038.20	11 812	87	136
19	1svr	81 198	1435	4666	10	−1731.54	7 294	53	138
20	1frd	81 972	1478	4387	9	−2890.28	5 676	41	139
21	1a2s	84 527	1272	4457	16	−1941.37	11 461	83	139
22	1neq	89 457	1187	4738	19	−1756.02	15 077	106	142
23	1a63	132 134	2065	7003	11	−2404.07	19 804	139	143
24	1a7m	147 121	2809	7769	54	−2184.05	124 326	852	146

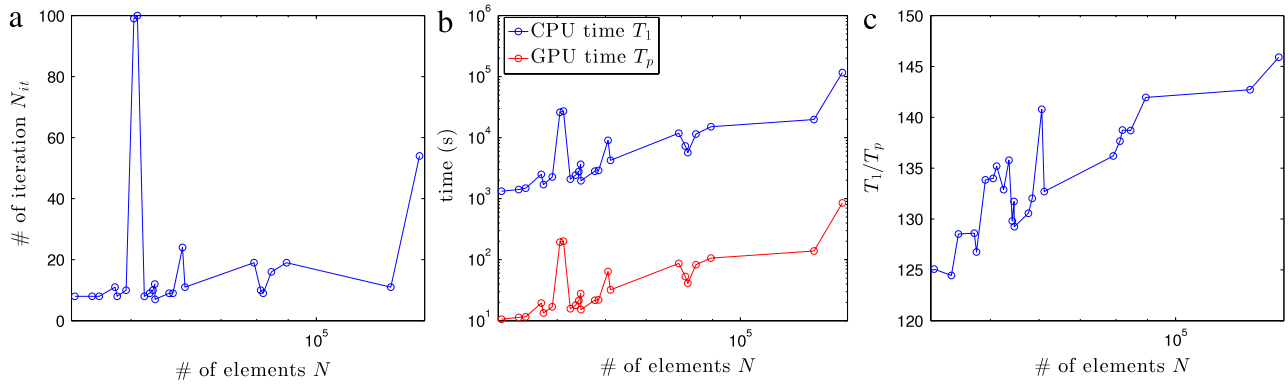

Fig. 2. Numerical results for solving PB equation and computing electrostatic solvation energy on a set of 24 proteins: (a) number of iterations; (b) CPU time T_1 and GPU time T_p; (c) the parallel speedup T_1/T_p.

Table 4
Numerical results for solving PB equation and computing the electrostatic solvation energy on protein 1frd and 1svr.

Density	1frd				1svr			
	# of ele.	E_{sol}	GPU T_p (s)	# of it.	# of ele.	E_{sol}	GPU T_p(s)	# of it.
1	10 176	−3360.69	2	18	13 974	−2038.30	2	11
2	17 710	−3003.45	2	9	22 946	−1833.41	4	10
4	34 520	−2931.05	8	9	38 116	−1769.44	10	9
8	66 294	−2894.02	28	9	71 030	−1735.37	31	9
16	134 180	−2880.90	109	9	144 168	−1723.64	140	10
32	266 702	−2872.65	423	9	284 478	−1716.94	642	11

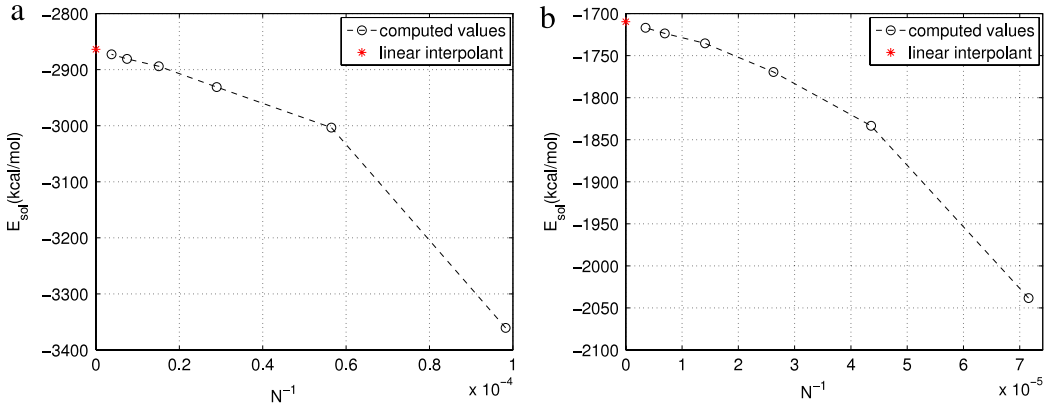

Fig. 3. Accuracy tests in terms for electrostatic solvation energy for proteins 1frd (a) and 1svr (b).

performs numerical integration with centroid collocation schemes. The numerical tests on spherical cavities show that GABI-PB solver can achieve 0.5th order convergence on surface potentials relative to the number of elements, which is a 1st order convergence relative to the length. The accurate surface potentials are of vital importance to molecular modelings that are sensitive to electrostatics near or on the molecular surface. Meanwhile, this direct-sum boundary integral implementation uses memory efficiently. It has been shown that we only need to allocate altogether $6N_c + 11N$ size-of-double device memory, where N_c is the number of atoms and N is the number of triangular elements. In addition, numerical tests on a series of 24 proteins show fast convergence, consistent electrostatic solvation energy computation, as well as 120–150X speedup, using a single GPU (Nvidia Tesla M2070) card.

The major limitation of the direct-sum scheme is obviously the $\mathcal{O}(N^2)$ computational cost, which becomes prohibitively expensive when the dimension of the problem increases to certain level. Even though using multiple GPU cards can offset some of this effect, it costs extra design, implementation, and hardware purchase. A remedy to this is, applying the fast algorithms such as the $\mathcal{O}(N \log N)$ treecode [17] and the much more complicated and memory-consuming $\mathcal{O}(N)$ FMM [32], which is under our investigation. It is hard to deny that the adoption of fast algorithms is necessary for large sized problems. However, there is a region on which the direct-sum GABI-PB solver has advantages over boundary integral PB solvers with fast algorithms. This region is bounded by what we called critical point, where the fast algorithms surpass the direct sum. For example, the CPU implementation of a treecode algorithm in solving boundary integral PB equation [17] shows a critical point at about $N = 4000$ with $p = 3$ (the order of Taylor expansion) and MAC ≤ 0.5 (multipole acceptance criterion, the ratio between cluster radius and the distance between target particle and the center of the cluster). In GPU implementation, the critical point will be much bigger in considering the communication and memory access. We will identify the value of critical point in our future work.

Memory usage is a critical factor of GPU performance. For solving boundary integral PB, direct-sum uses about 1/3 of memory of treecode [17] and 1/7 of memory of FMM [32]. The current GPU implementation is run on an available Tesla M2070 card. When we are considering to rerun all the tests on a Tesla

M2090, Nvidia announced its release of Tesla K10 followed by K20 and K20x, showing the rapid hardware update year after year. Taking M2070 and K20x as examples for comparison, the peak double precision floating points performance increases from 515Gflops to 1.31Tflops, the memory bandwidth increases from 150 GB/s to 250 GB/s, and the number of CUDA cores increases from 448 to 2688. However the memory is still limited at 6Gbyte. These comparisons indicate the proposed direct-sum boundary integral PB solver, benefited from its low memory use, will demonstrate continuously improving performance with the flow of GPU hardware updates.

In addition to including fast algorithms, there are many spaces in which GABI-PB can be improved and extended. For example, we are looking for better triangulation programs for the molecular surfaces [36–38] to avoid the slow convergence pattern for some proteins as seen in our tests. A more challenging problem is the application of GABI-PB to molecular dynamics [39,40], where the PB equation will be solved at every time sampling. For molecular surfaces discretized within 50,000 elements, GABI-PB can resolve each sample in a few seconds or less. Furthermore, the GPU-accelerated boundary integral scheme has the potential to solve other integral equations such as the Helmholtz equation and Maxwell Equations.

Acknowledgments

The work was supported by NSF grant DMS-0915057, University of Alabama new faculty startup fund and Alabama Supercomputer Center. The authors thank Robert Krasny for helpful discussions.

References

[1] N.A. Baker, Improving implicit solvent simulations: a Poisson-centric view, Curr. Opin. Struct. Biol. 15 (2005) 137–143.
[2] F.M. Richards, Areas, volumes, packing and protein structure, Annu. Rev. Biophys. Bioeng. 6 (1977) 151–176.
[3] M.L. Connolly, Molecular surface triangulation, J. Appl. Crystallogr. 18 (1985) 499–505.
[4] J.G. Kirkwood, Theory of solution of molecules containing widely separated charges with special application to Zwitterions, J. Chem. Phys. 7 (1934) 351–361.
[5] M. Holst, F. Saied, Multigrid solution of the Poisson–Boltzmann equation, J. Comput. Chem. 14 (1993) 105–113.
[6] W. Rocchia, E. Alexov, B. Honig, Extending the applicability of the nonlinear Poisson–Boltzmann equation: multiple dielectric constants and multivalent ions, J. Phys. Chem. B 105 (2001) 6507–6514.
[7] W. Im, D. Beglov, B. Roux, Continuum solvation model: computtation of electrostatic forces from numberical solutions to the Poisson–Boltzmann equation, Comput. Phys. Comm. 111 (1998) 59–75.
[8] R. Luo, L. David, M.K. Gilson, Accelerated Poisson–Boltzmann calculations for static and dynamic systems, J. Comput. Chem. 23 (2002) 1244–1253.
[9] N.A. Baker, D. Sept, M.J. Holst, J.A. McCammon, The adaptive multilevel finite element solution of the Poisson–Boltzmann equation on massively parallel computers, IBM J. Res. Dev. 45 (2001) 427–438.
[10] Z.H. Qiao, Z.L. Li, T. Tang, A finite difference scheme for solving the nonlinear Poisson–Boltzmann equation modeling charged spheres, J. Comput. Math. 24 (2006) 252–264.
[11] D. Chen, Z. Chen, C.J. Chen, W.H. Geng, G.W. Wei, MIBPB: a software package for electrostatic analysis, J. Comput. Chem. 32 (2011) 756–770.
[12] 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.
[13] A. Boschitsch, M. Fenley, H.-X. Zhou, Fast boundary element method for the linear Poisson–Boltzmann equation, J. Phys. Chem. B 106 (2002) 2741–2754.
[14] B.Z. Lu, X.L. Cheng, J.F. Huang, J.A. McCammon, Order N algorithm for computation of electrostatic interactions in biomolecular systems, Proc. Natl. Acad. Sci. USA 103 (2006) 19314–19319.
[15] R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada, Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUS and a billion unknowns, Comput. Phys. Comm. 182 (2011) 1272–1283.
[16] M.D. Altman, J.P. Bardhan, J.K. White, B. Tidor, Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized Poisson–Boltzmann equation with curved boundary elements, J. Comput. Chem. 30 (2009) 132–153.
[17] W.H. Geng, R. Krasny, A treecode-accelerated boundary integral Poisson–Boltzmann solver for electrostatics of solvated biomolecules, J. Comput. Phys. (2012) (submitted for publication).
[18] A. Bordner, G. Huber, Boundary element solution of the linear Poisson–Boltzmann equation and a multipole method for the rapid calculation of forces on macromolecules in solution, J. Comput. Chem. 24 (2003) 353–367.
[19] J. Liang, S. Subranmaniam, Computation of molecular electrostatics with boundary element methods, Biophys. J. 73 (1997) 1830–1841.
[20] Y.N. Vorobjev, H.A. Scheraga, A fast adaptive multigrid boundary element method for macromolecular electrostatic computations in a solvent, J. Comput. Chem. 18 (1997) 569–583.
[21] C. Bajaj, S.-C. Chen, A. Rand, An efficient higher-order fast multipole boundary element solution for Poisson–Boltzmann-based molecular electrostatics, SIAM J. Sci. Comput. 33 (2011) 826–848.
[22] A.W. Goetz, M.J. Williamson, D. Xu, D. Poole, S.L. Grand, R.C. Walker, Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized Born, J. Chem. Theory Comput. 8 (2012) 1542–1555.
[23] A.W. Goetz, R. Salomon-Ferrer, D. Poole, S.L. Grand, R.C. Walker, Routine microsecond molecular dynamics simulations with AMBER—Part II: particle Mesh Ewald, (2013) (in preparation).
[24] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (2005) 1781–1802.
[25] J.E. Stone, D.J. Hardy, I.S. Ufimtsev, K. Schulten, GPU-accelerated molecular modeling coming of age, J. Mol. Graph. Modell. 29 (2010) 116–125.
[26] L. Nyland, M. Harris, J. Prins, Fast N-body simulation with CUDA, in: GPU Gem, vol. 3, 2009, pp. 677–695 (Chapter 31).
[27] M. Burtscher, K. Pingali, An efficient CUDA implementation of the tree-based Barnes Hut n-body algorithm, in: GPU Computing Gems Emerald Edition, 2011, pp. 75–92 (Chapter 6).
[28] J. Barnes, P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, Nature 324 (1986) 446–449.
[29] R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada, Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns, Comput. Phys. Comm. 182 (2010) 1272–1283.
[30] B.J. Yong, A.M. Lenhoff, A boundary element method for molecular electrostatics with electrolyte effects, J. Comput. Chem. 11 (1990) 1080–1086.
[31] M.F. Sanner, A.J. Olson, J.C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.
[32] B.Z. Lu, X.L. Cheng, J.A. McCammon, New-version-fast-multipole-method accelerated electrostatic calculations in biomolecular systems, J. Comput. Phys. 226 (2007) 1348–1366.
[33] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–859.
[34] A.D. MacKerell Jr., D. Bashford, M. Bellott, J.D. Dunbrack, M.J. Evanseck, M.J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F.T.K. Lau, C. Mattos, S. Michnick, T. Ngo, D.T. Nguyen, B. Prodhom, W.E. Reiher, B. Roux, M. Schlenkrich, J.C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, M. Karplus, All-atom empirical potential for molecular modeling and dynamics studies of proteins, J. Phys. Chem. 102 (1998) 3586–3616.
[35] W.H. Geng, S.N. Yu, G.W. Wei, Treatment of charge singularities in the implicit solvent models, J. Chem. Phys. 128 (2007) 114106.
[36] P. Bates, G.W. Wei, S. Zhao, Minimal molecular surfaces and their applications, J. Comput. Chem. 29 (2008) 380–391.
[37] M.X. Chen, B.Z. Lu, TMSmesh: a robust method for molecular surface mesh generation using a trace technique, J. Chem. Theory Comput. 7 (2011) 203–212.
[38] D. Xu, Y. Zhang, Generating triangulated macromolecular surfaces by Euclidean distance transform, PLoS One 4 (12) (2013) e8140. http://dx.doi.org/10.1371/journal.pone.0008140.
[39] W.H. Geng, G.W. Wei, Multiscale molecular dynamics using the matched interface and boundary method, J. Comput. Phys. 230 (2011) 435–457.
[40] Q. Lu, R. Luo, A Poisson–Boltzmann dynamics method with nonperiodic boundary condition, J. Chem. Phys. 119 (2003) 11035–11047.