Computer Physics Communications 190 (2015) 173–181

Contents lists available at ScienceDirect

Computer Physics Communications

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

Parallel AFMPB solver with automatic surface meshing for calculation of molecular solvation free energy☆

Bo Zhang [a], Bo Peng [b], Jingfang Huang [c], Nikos P. Pitsianis [d,e], Xiaobai Sun [e], Benzhuo Lu [b,*]

[a] Center for Research in Extreme Scale Technologies, Indiana University, IN, USA

[b] State Key Laboratory of Scientific/Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

[c] Department of Mathematics, the University of North Carolina at Chapel Hill, NC, USA

[d] Department of Electrical and Computer Engineering, Aristotle University, Greece

[e] Department of Computer Science, Duke University, NC, USA

ARTICLE INFO

Article history:
Received 11 August 2014
Received in revised form
23 November 2014
Accepted 25 December 2014
Available online 23 January 2015

Keywords:
Poisson–Boltzmann equation
Boundary integral equation
Automatic surface meshing
Solvation free energy
Fast multipole methods
Parallelization
Cilk Plus

ABSTRACT

We present PAFMPB, an updated and parallel version of the AFMPB software package for fast calculation of molecular solvation-free energy. The new version has the following new features: (1) The adaptive fast multipole method and the boundary element methods are parallelized; (2) A tool is embedded for automatic molecular VDW/SAS surface mesh generation, leaving the requirement for a mesh file at input optional; (3) The package provides fast calculation of the total solvation-free energy, including the PB electrostatic and nonpolar interaction contributions. PAFMPB is implemented in C and Fortran programming languages, with the Cilk Plus extension to harness the computing power of both multicore and vector processing. Computational experiments demonstrate the successful application of PAFMPB to the calculation of the PB potential on a dengue virus system with more than one million atoms and a mesh with approximately 20 million triangles.

Program summary

Program title: Parallel AFMPB

Catalogue identifier: AEGB_v2_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEGB_v2_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: GNU General Public License, version 2

No. of lines in distributed program, including test data, etc.: 40558

No. of bytes in distributed program, including test data, etc.: 2349976

Distribution format: tar.gz

Programming language: Mixed C and Fortran, Compiler: Intel or GNU with Cilk Plus enabled.

Computer: Any, but the code is mainly designed for multicore architectures.

Operating system: Linux.

RAM: Depends on the size of the discretized biomolecular system.

Classification: 3.

Catalogue identifier of previous version: AEGB_v1_1

Journal reference of previous version: Comput. Phys. Comm. 184 (2013) 2618

External routines: Users are allowed to use external routines/libraries (e.g., MSMS [6] and TMSMesh [4]) to generate compatible surface mesh input data if they choose not to use the embedded automatic mesh generation tool in the package. Post-processing tools such as VCMM [5] and VMD [3] can also be used for visualization and analyzing results.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: bzlu@lsec.cc.ac.cn (B. Lu).

http://dx.doi.org/10.1016/j.cpc.2014.12.022

The package uses two subprograms: (1) The iterative Krylov subspace solver, SPARSKIT, from Yousef Saad [2]; and (2) Cilk-based parallel fast multipole methods from FMMSuite [1].

Does the new version supersede the previous version? Yes

Nature of problem: Numerical solution of the linearized Poisson–Boltzmann equation that describes electrostatic interactions of molecular systems in ionic solutions.

Solution method: The linearized Poisson–Boltzmann equation is reformulated as a boundary integral equation and is subsequently discretized using the node-patch scheme. The resulting linear system is solved using Krylov subspace solvers iteratively. The reformulation of the equation provides an upper bound for the number of iterations. Within each iteration, the matrix–vector multiplication is accelerated using the adaptive plane-wave expansion based fast multipole methods. The majority of the codes are parallelized using the Cilk runtime.

Reasons for new version: New functions are added and a few old functions like force calculations are removed. The algorithm is parallelized and most parts of the code are rewritten.

Summary of revisions: The computation is parallelized and an automatic mesh generation method for BEM is added.

Restrictions: The program has only been tested on machines running Linux operating system.

Additional comments: The Cilk runtime used in the development and testing is from the Intel compiler Suite. The GNU Cilk Plus and Cilk Plus/LLVM branches have not been tested.

Running time: The running time depends on the number of discretized elements (N) and their distribution. It also depends on the number of cores used in the computation.

References:

[1] http://www.fastmultipole.org/.

[2] http://www-users.cs.umn.edu/~saad/software/.

[3] http://www.ks.uiuc.edu/Research/vmd/.

[4] http://www.continuummodel.org.

[5] S. Bai, B. Lu, VCMM: A visual tool for continuum molecular modeling. J. Mol. Graph. Model. 50 (2014) 44–49.

[6] Scanner, F. Michel, Olson, J. Arthur, Spehner, J. Claude, Reduced surface: An efficient way to compute molecular surfaces. Biopolymers 38 (1996) 305–320.

1. Introduction

The Poisson–Boltzmann (PB) equation has been widely used to model the electrostatic properties of biomolecules, membranes, colloids, polyelectrolytes, and macromolecular objects. Efficient numerical solution of PB equations has been an active research topic in scientific computing. Many algorithms have been introduced, including the multigrid method [1], precorrected-FFT method [2], tree code [3], and the fast multipole method (FMM) [4–7]. These algorithms reduce the computational complexity from $O(N^2)$ to $O(N \log N)$, or asymptotically optimal $O(N)$, where N is the problem size. However, the efficiency was not satisfactory when the PB calculation was applied to the macromolecular systems or an ensemble of molecular structures.

Substantial progress has been made in improving the efficiency of PB calculations, such as the use of many-core graphic processing units (GPUs) accelerators [8,9], distributed and shared memory programming via MPI and OpenMP [1,10–12]. In the work by Geng et al. [10], a parallel higher-order boundary integral equation method is implemented using MPI.

In this paper, we present a parallel version of the adaptive fast multipole Poisson–Boltzmann equation solver (PAFMPB). Significant differences from the previous sequential AFMPB package [6] are: (1) PAFMPB targets multicore computer architectures and utilizes Cilk Plus [13] for parallelization. (2) PAFMPB adds an integrated automatic mesh generation option to handle the circumstance when an external mesh file is not easily obtainable or the mesh quality is not satisfactory for numerical simulation. (3) The computation of the total solvation-free energy contains both polar and nonpolar energy terms. (4) PAFMPB is written in mixed C and Fortran programming languages. In preliminary numerical experiments with PAFMPB, we have observed substantial improvements in terms of parallel efficiency and memory usage.

The rest of the paper is organized as follows. In Section 2, we briefly describe the individual elements of the PAFMPB package, namely, the PB model, the boundary integral equation (BIE) reformulation, the surface mesh generation method, the node-patch discretization approach, the Krylov subspace method, fast multipole method with plane-wave expansion, and the Cilk Plus system for parallelization. We present the code structure in Section 3. Preliminary numerical results are presented in Section 4 to demonstrate the accuracy and parallel performance of the solver. We conclude the paper in Section 5.

2. Mathematical models and discretization methods

We first give a brief review of the mathematical theories and computational methods that underlie the PAFMPB package. We then detail the new features in PAFMPB, in addition to that in [6] and references therein.

2.1. Continuum electrostatic models and boundary integral equations

Electrostatic interactions within and between molecules in electrolyte solution have long been recognized as playing a variety of important roles in determining molecular structure and binding

activity. Consider a molecule immersed in an electrolyte solution. The interior region (molecule) and the exterior region (solvent) are denoted by Ω_1 and Ω_2, respectively. The interior potential $\phi^{\text{int}}(\mathbf{r})$ is governed by the Poisson equation

$$-\nabla\cdot(\epsilon_{\text{int}}\nabla\phi^{\text{int}}(\mathbf{r})) = \sum_{i=1}^{M} q_i\delta(\mathbf{r},\mathbf{c}_i) \quad \text{for } \mathbf{r}\in\Omega_1. \tag{1}$$

The exterior potential $\phi^{\text{ext}}(\mathbf{r})$ is governed by the Poisson–Boltzmann equation [14]. In this work, we use an approximated form of the PB equation, specifically, the linearized PB (LPB) equation as follows:

$$-\nabla\cdot(\epsilon_{\text{ext}}\nabla\phi^{\text{ext}}(\mathbf{r})) = -\kappa^2\phi^{\text{ext}}(\mathbf{r}) \quad \text{for } \mathbf{r}\in\Omega_2. \tag{2}$$

Here, ϵ_{int} denotes the dielectric constant inside the molecule, and the molecule (the right-hand side of Eq. (1)) is represented by M point charges q_ie_c at positions $\mathbf{c}_i$, $i = 1, 2, \ldots, M$. The reciprocal of the Debye length $1/\kappa$ characterizes the screening effect due to the surrounding electrolyte solution. At molecular surface Γ, the potential and the normal component of the electric displacement must be continuous,

$$\phi^{\text{int}} = \phi^{\text{ext}} \quad \text{and} \quad \epsilon_{\text{int}}\frac{\partial\phi^{\text{int}}}{\partial n} = \epsilon_{\text{ext}}\frac{\partial\phi^{\text{ext}}}{\partial n}, \tag{3}$$

where n denotes the outward unit normal vector to the molecular surface and ϵ_{ext} is the dielectric constant of the electrolyte solution.

Both equations in (1) and (2) can be recast into boundary integral equations (BIEs) using Green's second identity. In fact, a well-conditioned second kind Fredholm integral equation can be obtained through a combination of the BIEs and derivative forms of the BIEs [6]:

$$\begin{aligned}\left(\frac{1}{2\epsilon}+\frac{1}{2}\right)f_p &= \oint_S\left[(G_{pt}-u_{pt})h_t - \left(\frac{1}{\epsilon}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)f_t\right]dS + \frac{1}{\epsilon_{\text{ext}}}\sum_k q_kG_{pk}\\ \left(\frac{1}{2\epsilon}+\frac{1}{2}\right)h_p &= \oint_S\left[\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\epsilon}\frac{\partial u_{pt}}{\partial n_0}\right)h_t - \frac{1}{\epsilon}\left(\frac{\partial^2 G_{pt}}{\partial n_0\partial n}-\frac{\partial^2 u_{pt}}{\partial n_0\partial n}\right)f_t\right]dS + \frac{1}{\epsilon_{\text{ext}}}\sum_k q_k\frac{\partial G_{pk}}{\partial n_0}.\end{aligned} \tag{4}$$

Here, $\epsilon = \frac{\epsilon_{\text{ext}}}{\epsilon_{\text{int}}}$, n_0 and n are the unit normal vectors at point $p\in\Gamma$ and $t\in\Gamma$, respectively. We denote by f_p and h_p the values of ϕ^{ext} and $\frac{\partial\phi^{\text{ext}}}{\partial n}$ at point p, respectively. We make use of the fundamental solutions to the Poisson equation G_{pt} and the LPB equation u_{pt},

$$G_{pt} = \frac{1}{4\pi|\mathbf{r}_t-\mathbf{r}_p|}, \qquad u_{pt} = \frac{\exp(-\kappa|\mathbf{r}_t-\mathbf{r}_p|)}{4\pi|\mathbf{r}_t-\mathbf{r}_p|}.$$

2.2. Molecular surfaces and mesh generation

Numerical solution of the BIEs in Eq. (4) requires a discretization of the molecular surfaces, namely, a mesh. The previous version of AFMPB requires a mesh file as an input file. The mesh may be generated using existing packages such as MSMS [15] (for solvent excluded surface (SES) mesh generation). The mesh generation may be considered as a preprocessing step, separated from the AFMPB execution. Besides the inconvenience, some inconsistency may occur between the mesh generation and the AFMPB requirement. In PAFMPB, in addition to accepting external mesh files as input, we provide an embedded meshing procedure that can discretize both the molecular van der Waals (VDW) surface and solvent-accessible surface (SAS) into triangular or quadrilateral patches, which are to be used directly in the node-patch discretization discussed in the next section. For large molecules, we make use of the recently developed TMSmesh package for generating triangular surface mesh for the molecular Gaussian surface [16], which is smooth and analytically differentiable with respect to the position of atoms. In the rest of this section, we describe the automatic mesh generation procedure for VDW/SAS surface mesh and TMSmesh [17,16] for Gaussian surface mesh generation, respectively.

Different definitions of the molecular surface (the boundary between the solute and the solvent) can affect the PB calculation results. Zhou et al. [18] compared the VDW surface and molecular SES when applied in the PB calculations and concluded that the VDW surface may yield better results than the molecular SES.

2.2.1. An auto-meshing procedure

In PAFMPB, an auto-meshing option is provided to conveniently generate meshes for either the SAS or VDW molecular surfaces. The latitude–longitude based method discretizes the surface into either triangular or quadrilateral patches. These patches can be directly used in the node-patch BEM [19], which needs only the normal, area, and centroid coordinates for each node patch. The method shares similar features as the program developed by Lee and Richard [20], including the estimate of the SAS area and static accessibility.

Following the convention in the study of biomolecular systems, we represent a molecule as a set of overlapping spheres, and define the VDW or SAS surface as the topological boundary of the spheres as follows: the set of atomic spheres is denoted as $\{O_1, O_2, \ldots, O_M\}$, each with radius $\rho_i = R_i^{\text{VDW}} + R_{\text{p}}$, where R_i^{VDW} is the VDW radius, and R_{p} is a user-controllable parameter for different types of surfaces. When $R_{\text{p}} = 0$, the surface is the VDW surface and when $R_{\text{p}} = R_{\text{water}} = 1.4$ Å, the surface is the SAS surface.

Our mesh generation subroutine consists of two stages. In the first stage, each spherical surface is discretized using suitable latitude and longitude lines to form surface elements and their areas and normal directions are calculated. For each sphere O_i, the spherical surface is discretized by dividing the azimuthal angle β (from 0 to 2π) into n equispaced intervals, $0 = \beta_0 < \beta_1 < \cdots < \beta_{n-1} < 2\pi$, and dividing the sine of the polar angle θ (from -1 to 1) into m equispaced intervals, $-\frac{\pi}{2} = \theta_0 < \theta_1 < \cdots < \theta_m = \frac{\pi}{2}$. We define $[\beta_j, \beta_{j+1}]\times[\theta_k, \theta_{k+1}]$ as one element. Its area is $\rho_i^2(\sin\theta_{k+1} - \sin\theta_k)(\beta_{j+1}-\beta_j)$, and its normal direction is

$$\left(\cos\frac{\theta_k+\theta_{k+1}}{2}\cos\frac{\beta_j+\beta_{j+1}}{2}, \cos\frac{\theta_k+\theta_{k+1}}{2}\sin\frac{\beta_j+\beta_{j+1}}{2}, \sin\frac{\theta_k+\theta_{k+1}}{2}\right).$$

The surface discretization in this step generates l elements $e_{i,1}$, $\ldots$, $e_{i,l}$ for sphere i, $l = mn$.

In the second stage, we modify any element whose central point is located *inside* any sphere(s). Such elements are referred to as "buried elements". For each O_i, we find all its intersecting sphere(s), and then check whether each element $e_{i,j}$ is a buried element, $1 \leqslant j \leqslant l$. Non-buried elements are added to the surface patch list directly. A two-atom system using the above method is illustrated in Fig. 2.

This algorithm is simple, but has quadratic complexity in M, the number of patches. We thereby recommend this automatic meshing option only for modest size molecules. For macromolecules, external mesh generation tools should be applied, as in the original AFMPB package. Particularly, we recommend the recently developed TMSmesh package for generating macromolecular surface meshes for the BIE formulation.

Efficient generation of a high-quality mesh has been a challenging problem to the scientific computing community for decades.

Mesh generation remains a major bottleneck for computationally intensive electrostatic analysis. To further improve efficiency, we are currently investigating (1) octree-structure based schemes to reduce the complexity of the latitude–longitude based method; (2) proper parameterization procedure and higher order geometric representation to reduce the number of elements; and (3) fast mesh modification strategies when the shape of the molecule changes slightly at each time step. New results will be incorporated in future releases of the PAFMPB package.

2.2.2. TMSmesh and Gaussian molecular surface

For a macromolecular system, it has been a long-standing challenge to generate a high-quality mesh. We recently developed a robust program, *TMSmesh* [16], for generating macromolecular Gaussian surfaces and the corresponding meshes.

The Gaussian molecular surface is defined as a level set of the summation of the Gaussian kernel functions as follows:

$$\Phi(\mathbf{x}) = \sum_{i=1}^{M} e^{-d(|\mathbf{x}-\mathbf{c}_i|^2-\rho_i^2)} = 1, \quad \mathbf{x} \in \mathbb{R}^3, \tag{5}$$

where $\mathbf{c}_i$ and ρ_i are the location and radius of atom i, respectively, and d is a positive parameter controlling the decay speed of the kernel functions. Compared to the VDW and SAS surfaces, the Gaussian surface is usually smooth, without corner or edge singularities. In addition, it is possible to adjust the parameter d to approximate other types of surfaces.

In TMSmesh, a trace technique is used to mesh the Gaussian surface. The program has a linear computational complexity in the number of atoms, and is capable of treating arbitrarily large macromolecules. An example of Gaussian surface mesh for the dengue virus system consisting of 1082160 atoms is shown in Fig. 3(a), with 9758 426 vertices and 19 502 784 elements. TMSmesh has also been used to stably generate meshes for other systems such as the 70S ribosome [16]. The mesh quality has been validated by comparisons with meshes from other programs, and by the simulation results for the BIE-based new version of the AFMPB solver. Further details of the algorithms and comparisons can be found in [16].

2.3. Discretization and the "node-patch" method

We adopt a "node-patch" approach to discretize the BIEs in Eq. (4). The approach can be considered as a modified linear element method in accuracy but it takes advantage of the ease of programming as for the constant element method. Using the generated closed mesh approximation of the molecular boundary Γ, the unknowns in the traditional linear element method are defined at the vertices, and the surface function $f(\mathbf{r})$, $\mathbf{r} \in \Gamma$, is approximated by $f(\mathbf{r}) = \sum_{k=1}^{V} f_k N_k(\mathbf{r})$ using the linear element basis functions $\{N_k(\mathbf{r}),\ k = 1, 2, \ldots, V\}$, where V is the number of vertices in the mesh of Γ, f_k is the value of $f(\mathbf{r})$ at vertex k, and $N_k(\mathbf{r})$ is a piecewise linear function which equals 1 at node k and 0 at all other nodes. Similarly, we have $h(\mathbf{r}) = \sum_{k=1}^{V} h_k N_k(\mathbf{r})$. The discretized form of Eq. (4) becomes

$$\begin{aligned}
&\left(\frac{1}{2\epsilon}+\frac{1}{2}\right) f_p = \frac{1}{\epsilon_{\text{ext}}} \sum_k q_k G_{pk} \\
&+ \sum_{k=1}^{V} \sum_{t\in\varphi(k)} \left[h_k \int_{S_t} \left(G_{pt} - u_{pt}\right) N_k dS_t \right. \\
&\left. - f_k \int_{S_t} \left(\frac{1}{\epsilon} \frac{\partial G_{pt}}{\partial n} - \frac{\partial u_{pt}}{\partial n} \right) N_k dS_t \right] \\
&\left(\frac{1}{2}+\frac{1}{2\epsilon}\right) h_p = \frac{1}{\epsilon_{\text{ext}}} \sum_k q_k \frac{\partial G_{pk}}{\partial n_0} \\
&+ \sum_{k=1}^{V} \sum_{t\in\varphi(k)} h_k \int_{S_t} \left(\frac{\partial G_{pt}}{\partial n_0} - \frac{1}{\epsilon} \frac{\partial u_{pt}}{\partial n_0} \right) N_k dS_t \\
&- \sum_{k=1}^{V} \sum_{t\in\varphi(k)} f_k \int_{S_t} \frac{1}{\epsilon} \left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n} - \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} \right) N_k dS_t
\end{aligned} \tag{6}$$

where $\varphi(k)$ is the set of indices for surface elements that contain vertex k on the mesh for Γ. Notice that in Eq. (6), one needs to compute the nearly-singular and singular integrals representing the convolution of Green's function (and its derivatives) with a *linear* function. For efficiency, in the "node-patch" BEM approach [19], a patch is formed around each vertex (node), and the function is assumed to be a constant on the node-patch, namely, the function on each element is a piecewise constant (non-continuous) function. When the node-patch is carefully designed, one should expect comparable accuracy to that with the linear element methods. One needs to compute only the convolution of the Green's function with a piecewise *constant* function. The node-patch discretization leads to a linear system $Ax = b$, where x denotes the unknowns, A and b are the coefficient matrix and right hand side of Eq. (6), respectively.

The node-patch approach has several advantages. (1) The total number of unknowns is reduced for a prescribed accuracy requirement when compared with the constant element method, hence reducing the computational cost when solving the resulting linear system. (2) When the matrix is split into the near and far fields $A = A_{\text{near}} + A_{\text{far}}$, the calculation and storage process of A_{near} by the node-patch approach are more economic compared with the linear element approach. (3) The source and target points are the one and same set of nodes, like the constant element method, allowing the application of the symmetric FMM from the FMMSuite package [19].

2.4. The Krylov subspace method for solving a linear system

For the discretized linear system, the PAFMPB solver searches for the optimal least-squares approximation of the solution in the Krylov subspace $K_m(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ for $m > 1$, where r_0 is the initial residual. As the resulting linear system is in general nonsymmetric, the GMRES, biconjugate gradient stabilized (BiCGStab), and transpose-free Quasi-Minimum Residual (TFQMR) can be applied. In the current version of PAFMPB, the subroutines from the open source package SPARSKIT [21] are used, which follow the "reverse communication protocol" with a simple and effective interface for integrating the iterative subroutines with a user-customized matrix–vector product procedure.

The convergence of a Krylov subspace method depends on the initial guess and eigenvalue distribution of the linear system. In the current implementation, the right hand side is used as the initial guess, as the formulation is already a Fredholm second kind equation. We are investigating possible approaches to further improving the efficiency of the Krylov iterative approach, by numerically preconditioning the linear system resulting from the Fredholm second-kind integral equation formulation, and by further exploiting the parallel potential of the Krylov subspace method.

2.5. The fast multipole method with plane wave expansion

Each step of the Krylov subspace method involves a matrix–vector multiplication, for which the FMM takes $O(N)$ operations while the direct evaluation takes $O(N^2)$ operations, where N is the dimension of linear system. In the PAFMPB package, we apply the new version of the fast multipole method [22], which lowers the time and space complexity further by a significant factor.

The use of plane wave expansions is the main difference between the original version of the FMM and its new version. The FMM arrives at the linear-scaling, asymptotically optimal complexity by exploiting the mathematical properties with novel computation mechanisms. The basic observation is that the submatrix corresponding to the interactions between sources and targets that are sufficiently far apart is of low rank, bounded from above by a constant. The basic idea is to partition the interacting particles into far and near-neighbors, and partition them recursively at multiple spatial scale levels. The FMM has a simple spatial partition scheme, which is described in a data structure such as an octree for the 3D case. The root node represents a box enclosing all the source and target particles. The root box is partitioned in halves in each dimension into eight children boxes. Each child box is partitioned the same way, recursively. The far–neighbor particle interactions are then orchestrated into the interactions between particle ensembles in far–neighbor boxes at multiple levels. Each submatrix corresponding to the particle interactions between a pair of far–neighbor boxes is of lower rank. The information in each source box is aggregated up to the parent box, passed across at the same level to designated far–neighbor boxes, according to the interaction lists. The accumulated potential at each target box is disaggregated down to its children boxes. In its original version, the FMM represents and compresses the entire interaction matrix in terms of translation matrices via the classical spherical harmonics expansions [23]. In its new version [22], the FMM uses the plane wave or exponential expansions to diagonalize the up, down and across translations, and hence further reduces the translation and interaction complexity.

2.6. Cilk Plus and PAFMPB

A major change in PAFMPB package lies in parallelizing the FMM and BEM components to utilize multicore computer architectures. Here, parallelizing the algorithms can be considered as traversing the directed acyclic graph (DAG) of the computation dependencies in parallel. We point out that the problem of parallel traversing a DAG of N nodes in the shortest time, with the number of processing units constant (more than two) is NP-complete [24]. We study strategies for near-optimal solutions.

An earlier strategy studied was reported in [25], based on a dynamic prioritization scheme, implemented using Pthreads. The idea is to rank the nodes in the DAG adaptively when the number of nodes is far more than the number of available processing units. Particularly, the ranking is based on the longest path of each node to the final sink node. When two frontier nodes are of the same rank, an ordered pair is associated with each node c, defined as $\langle d_m^-(c), n_s(c)\rangle$,

$$d_m^-(c) = \min_{(c,d)\in\mathcal{E}(G)} \deg^-(d), \tag{7}$$

where (c, d) is a directed edge from c to d in the edge set $\mathcal{E}(G)$, and $\deg^-(d)$ is the in-degree at node d. When $d_m^-(c) = 1$, $n_s(c)$ equals the number of direct successors of c with in-degree 1. That is, c is the only node in the way to release and enable these successor nodes. Otherwise, $n_s(c)$ equals the out-degree of c, $\deg^+(c)$. The completion of c makes its successors one step closer to the front. For two equal-ranked nodes u and v, we give u a higher priority if

$$\langle d_m^-(u), -n_s(u)\rangle < \langle d_m^-(v), -n_s(v)\rangle \tag{8}$$

in the lexicographical order. If the two nodes have the same priority, a random selection is applied to break the tie. Essentially, the dynamic prioritization approach implemented in [25] can be viewed as a work-sharing approach, where available tasks are put in a priority queue based on their rank for available processing units. Notice that mutual exclusion is needed for operations on the queue and this could become a bottleneck when the number of processing units increases.

More recently, we adopted an alternative approach to parallelizing the traversal of FMM DAG and the BEM components using Cilk Plus. Cilk Plus is an extension to the C/C++ language. It introduces three new keywords to express data and task parallelism. New tasks can be spawned using `cilk_spawn` and `cilk_for` and synchronized using `cilk_sync`. The runtime scheduler of Cilk Plus creates a worker thread for each processor core present on a system. Each worker maintains a double-ended queue (deque) for storing tasks yet to be executed. When a worker thread executes a `cilk_spawn` or `cilk_for` statement, it simply pushes new tasks onto the tail of its deque. Unlike the work-sharing approach in [25], the scheduler of the Cilk Plus runtime implements a work-stealing scheme with a proven performance guarantee [13]. When a worker has no work to do, it steals a task from the front of some other worker's deque. The Cilk Plus compiler and runtime is supported in the Intel compiler toolchain as well as in the open source compilers from the GCC and the LLVM. Our experiments show that Cilk Plus-based parallel FMM solvers are easy to manage and perform better than the dynamic prioritization code using Pthreads.

2.7. Evaluating the total solvation-free energy

In PAFMPB, an empirical approach is provided to compute the nonpolar contribution to the solvation free energy, which depends on the molecular surface area and volume. The related parameters are user-specified in the input file. Combining the polar (electrostatic) and nonpolar contributions, the total free energy can be computed as

$$\Delta E = \Delta E_{\mathrm{p}} + \Delta E_{\mathrm{np}}. \tag{9}$$

The nonpolar term, ΔE_{np}, includes the energetic cost of cavity formation, solvent rearrangement and solute–solvent dispersion interactions introduced when the uncharged solute is brought from vacuum into the solvent environment. The polar term ΔE_{p} is determined by the PB solution.

The solute charge generates a Coulombic potential field and polarizes the solvent, which in turn generates a reaction potential in the solute. The electrostatic potential of a molecule can therefore be expressed as the sum of the Coulomb potential ϕ_{c} and reaction field potential ϕ_{r} induced by solvent polarization:

$$\phi(\mathbf{r}) = \phi_{\mathrm{c}} + \phi_{\mathrm{r}}.$$

Once the electrostatic potential $\phi(\mathbf{r})$ is obtained, the electrostatic contribution of solvation-free energy is computed by

$$\begin{aligned}\Delta E_{\mathrm{p}} &= \frac{1}{2}\sum_{i=1}^{M} q_i \phi_{\mathrm{r}}(\mathbf{r}_i)\\ &= \frac{1}{2}\sum_{i=1}^{M} q_i \oint_S \left(G_{it}\frac{\partial \phi_t^{\mathrm{int}}}{\partial n} - \frac{\partial G_{it}}{\partial n}\phi_t^{\mathrm{int}}\right) dS_t\\ &= \frac{1}{2}\sum_{i=1}^{M} q_i \sum_t \left(\epsilon G_{it} h_t \Delta S_t - \frac{\partial G_{it}}{\partial n} f_t \Delta S_t\right).\end{aligned}$$

The nonpolar solvation energy depends on the surface area and volume:

$$\Delta E_{\mathrm{np}} = \gamma S + pV + b \tag{10}$$

where S and V are the surface area and volume of the cavity created by the molecule, respectively, and γ, p and b are fitted parameters, predefined in the input file. Parameter γ has the dimensions of a surface tension coefficient, parameter p has the pressure dimensions, and parameter b has the energy dimensions. They

depend on the force field and the definition of molecular surface because the atomic radii and charges (from the force field) and the surface definition determine together the boundary between the solvent and the solute and, therefore, the surface area S, volume V and the polar contributions to free energies. For instance, the particular values $\gamma = 0.005$ kcalmol^{-1}Å^{-2}, $p = 0.035$ kcalmol^{-1}Å^{-3} and $b = 0$ kcalmol^{-1} are used in [26]. In this paper, we omit the attractive nonpolar solvation interactions, for the same reason given in [26]. We mention that the nonpolar solvation energy expressions similar to Eq. (10) can be derived using the scaled particle theory [27]. A more detailed information for the evaluation of these terms is given in [28]. Eq. (10) reduces to the popular area-only nonpolar implicit solvent model when $p = 0$ [29,30]. In addition, b is set to 0 kcalmol^{-1} in all the following test examples of this work.

3. Code structure and implementation

PAFMPB features parallelized FMMs and dynamic memory allocation, making it possible to study protein systems with millions of atoms. In addition, an auto-discretization method for the VDW/SAS surfaces and an empirical formulation for calculating the nonpolar solvation energy are implemented to facilitate the use of the package. Another feature is the introduction of Linux command-line tools that can load and analyze the input file and accurately track the memory consumption.

In the following, we describe portability and installation of the package, several utility tools, file formats and I/O layers, and job running samples.

3.1. Portability and installation

PAFMPB is programmed mostly in ANSI C and Fortran 77. Memory allocation functions from the standard C library are used. After the user downloads the package, the following directories can be found upon extraction:

- `src`: contains the driver program `afmpb-main.c`, the interface file `afmpb-compute.c`, and utility file `afmpb-utils.c`. File `afmpb-solve.c` provides routines for BEM calculation. Files `fmm-graph.c`, `fmm-laplace.c`, `fmm-yukawa.c` and `fmm-utils.c` provide the parallelized FMM for Laplace and Yukawa kernels. File `afmpb-automesh.c` provides the auto mesh generation. File `sparskit.f` provides the Krylov subspace iterative solver [21].
- `include`: contains header files `adap_fmm.h` and `afmpb.h` that define data types and declare function prototypes.
- `example`: contains one `README` file, two sample job files (`job.sh` and `job-automesh.sh`), and the generated files (`input.txt` and `output.txt`).
- `tools`: contains mesh generation tools, including the TMSmesh tool. Details about these tools are provided in the next section.

In addition, a `Makefile` is supplied. The current package is based on Intel compiler and Linux command-line environments. The environment variable `LD_LIBRARY_PATH` for the library `libcilkrts.so` should be added before compiling and running PAFMPB.

3.2. Utility tools

The directory `tools` contains tools for mesh generation. One script (`pqrmsms.sh`) and two executable files (`MSMS, TMSmesh`) are provided. Given a PQR file (e.g. `fas2.pqr` in the directory), a MSMS format mesh can be generated by running the command

```
./pqrmsms.sh fas2.pqr 4 1.4
```

The last two parameters specify the node density (in the unit of 1 per Å^2) and probe radius in the unit of Å, respectively. One can also use TMSmesh (available at www.continuummodel.org) to generate an OFF format mesh by running the command

```
./TMSmesh fas2.pqr 0.6 2.0
```

The molecule and calculated surface potential can be visualized and analyzed using our visualization program VCMM [31], or the tool VMD as described in our previous version of AFMPB [6].

3.3. Job execution

PAFMPB is executed in a terminal environment, accepting the following parameters from the command line:

- `-i`: followed by input file name.
- `-o`: followed by output file name.
- `-s`: followed by surface potential file name.
- `-t`: followed by an integer number (the number of cores used).

PAFMPB expects the following input files: (a) a control parameter file (`input.txt` by default), (b) a PQR file containing atomic charges and their locations, and (c) a molecular surface mesh file when the auto meshing function is not used.

A log file (`output.txt` by default) and a surface potential file (default: `surfp.dat`) are generated automatically at the completion of PAFMPB execution.

The sample input files and job script files can be found in the program package and on our website www.continuummodel.org. A web server will also be provided.

4. Numerical results

In this section, we present numerical results for several test cases. In all cases, the relative dielectric constants are set to be 2 inside the molecule and 80 in the electrolyte solution, the monovalent ionic concentration is 150 mM, and the temperature is 300 K. Unless stated otherwise, N, M and N_{iter} denote the degree of freedom of the system (2 times the number of vertices), number of atoms in protein, and numbers of GMRES iterations for solving the LPB, respectively. The unit of the polar and nonpolar solvation energies is kcal/mol and the CPU time is measured in seconds.

4.1. Accuracy and efficiency of PAFMPB

We solve the LPB and compute the electrostatic solvation energy on a series of proteins with PAFMPB and previous versions of AFMPB. The numerical results are reported in Table 1. The first column is the molecule name. Columns 4 and 5 give the numbers of GMRES iterations for solving the LPB. We notice that both AFMPB solvers converge within 20 iterations, due to the well-conditioned integral equation formulation. Columns 6 and 7 report the polar solvation energies; results from different versions are very similar. The last two columns show the CPU time. Because of code optimization, the updated version shows better performance, especially for large systems.

4.2. Assessment of auto-generated mesh quality

For small molecules, PAFMPB works well using the automatically generated surface mesh. In this section, we first consider a spherical cavity. Since the analytical solution is available, to assess the accuracy of the auto-generated mesh, we measure the errors in the potential Φ (denoted by err_f) and in the normal derivative

Table 1
Computational performance on 5 proteins using a single core and using MSMS mesh: GLY, diALA, ADP, FasII, AChE. "old" and "new" denote the AFMPB and PAFMPB, respectively.

PQR	N	M	N_{iter}		E_{p}		CPU time	
			Old	New	Old	New	Old	New
GLY	1 640	7	5	6	−4.52	−4.52	0.26	0.22
diALA	2 030	20	6	7	−8.30	−8.30	0.39	0.35
ADP	5 660	39	7	8	−267.65	−267.80	1.31	1.14
FasII	36 702	906	11	11	−532.43	−532.43	17.57	15.27
AChE	427 020	8280	18	17	−2809.10	−2809.31	624.69	326.63
AChE	887 542	8280	19	18	−2818.75	−2819.50	1406.70	552.29
AChE	1335 670	8280	19	17	−2822.70	−2823.37	2027.20	892.38

Table 2
Accuracy from auto generated meshes and icosahedron-based meshes.

Auto generated meshes			Icosahedron-based meshes		
N	err_f	err_h	N	err_f	err_h
196	0.70	0.72	1 284	0.34	0.39
900	0.24	0.31	5 124	0.27	0.19
3 364	0.12	0.17	20 484	0.19	0.18
12 544	0.07	0.11	81 924	0.16	0.17

$\frac{\partial \Phi}{\partial n}$ (denoted by err_h). The error is defined as

$$\text{err}_f = \left(\int_\Gamma |f^{\text{num}} - f^{\text{exact}}|^2 dS \right)^{1/2}$$

where the superscript "num" and "exact" denote the numerical and exact solutions of f, respectively, and the definition of err_h is similar. The auto-generated mesh is compared with a set of more uniform icosahedron-based mesh (only used for spherical cavity) used in the old AFMPB. In our setup, $I = 150$ mM, $T = 300$ K, $f^{\text{exact}} = 3.69$, and $h^{\text{exact}} = -4.15$. Table 2 lists the errors at different auto-mesh densities and different icosahedron-based mesh densities. Table 2 shows that the auto-generated mesh leads to accurate results.

To further illustrate the quality of the auto-generated surface meshes on the electrostatic calculation for proteins, we compute the electrostatic solvation energies of GLY at various mesh resolutions. Since the exact solution is unavailable in this case, we compare the numerical iterations and solvation energies with those using MSMS meshes. Different probe radii (1.40, 0.10, 0.05, 0.00) and the same density 10 are used to generate four different MSMS mesh files. Table 3 shows the results from the auto-generated meshes and MSMS meshes. In nonpolar energy calculations, the surface tension coefficient γ and pressure p are set to be 0.005 kcalmol^{-1}Å^{-2} and 0.035 kcalmol^{-1}Å^{-3} [26], respectively.

As indicated by Table 3, the number of GMRES iterations using the auto-generated mesh remains stable, despite the increase of the automesh resolution. Furthermore, when the probe radius is smaller in MSMS (the molecular surface is then closer to VDW surface), the calculated energies are closer to those obtained using auto-meshing on the VDW surface.

Table 4
Memory and GMRES iteration steps for solving the LPBE using MSMS meshes on the case FasII. Memory is in unit of MB.

N	Memory	N_{iter}	E_{p}	E_{np}
67 192	1 395	47	−522.69	280.26
140 906	2 254	33	−525.77	280.39
280 744	3 305	13	−525.10	280.44
575 526	6 253	13	−524.81	280.47
1163 740	10 917	14	−524.03	281.50

4.3. Parallel efficiency

To study the speedup factor from parallelization, PAFMPB is tested on FasII, a 68-residue protein at various mesh resolutions. We list the CPU times and memory requirement using different number of cores. The simulations are performed on an eight-core desktop workstation with an Intel Xeon CPU @2.44 GHz and 48 GB memory. Table 4 displays the memory requirement and iterations. Fig. 1 shows the speedup for different cores.

In Table 4, the numbers of vertices, hence the mesh density, is approximately doubled each time. The solvation energies are approaching the value at the largest density. When the degrees of freedom (column 1) increase, the numbers of iterations (column 3) are not increasing. From Table 4 and Fig. 1(a), the memory and average CPU time per GMRES iteration scale almost linearly with the degrees of freedom. From Fig. 1(b), a parallel efficiency of more than 94% is achieved when using 4 cores and 86% when using 8 cores.

We present next, parallel scalability results with the AChE molecule data set. Table 5 displays the timing results in seconds. In this table, column 1 is the number of cores and column 2 is the time to compute the geometrical mesh information used in "node-patch" method. Column 3 is the time for computing the polar solvation energy by FMM calls. Column 4 is the time for computing the right hand side of Eq. (4). Column 5 is the time for assembling the matrix. Columns 6 and 7 are for the near-field and far-field (using FMM) matrix–vector products, respectively. The last column displays the total time for running the program. The most time-consuming part (more than three fourths of the total CPU time cost) is the far-field integration. The energy calculation, matrix assembly, and near-field integration also consume considerable CPU time. Most of these parts have good parallel scalability. The mesh

Table 3
Comparison of number of GMRES iterations and energy results from auto generated meshes (VDW surface) and MSMS meshes from different probe radius. R_p denotes probe radius.

Auto-generated meshes				MSMS meshes			
N	N_{iter}	E_{p}	E_{np}(VDW)	$R_{\text{p}}(N)$	N_{iter}	E_{p}	E_{np}
416	6	−4.47	1.40	1.40(820)	6	−4.52	2.93
876	7	−4.43	2.32	0.10(947)	8	−4.46	2.88
1762	7	−4.38	2.45	0.05(965)	6	−4.42	2.88
3529	7	−4.32	2.97	0.00(825)	6	−4.34	2.94

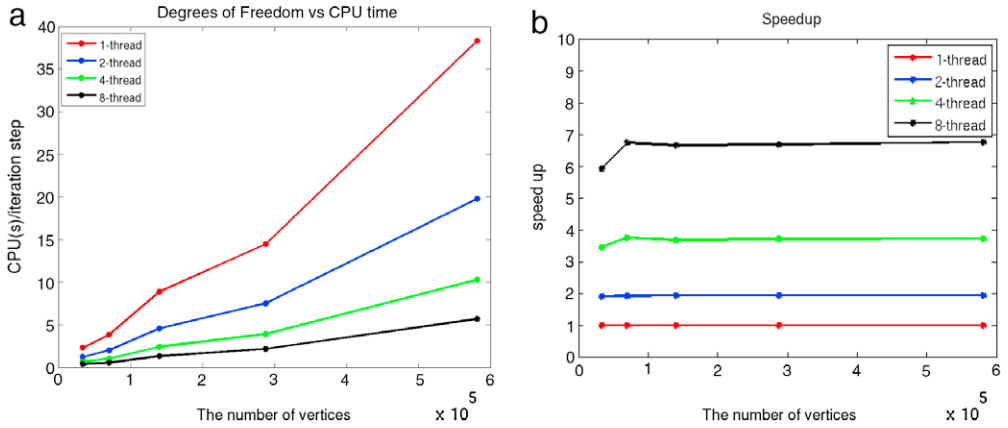

Fig. 1. Parallel performance of PAFMPB applied to molecule FasII. (a) Total CPU time for one GMRES iteration step as a function of mesh size. (b) Speedup for different number of cores using different mesh sizes (using the performance of one core as a reference).

Table 5
CPU times for main functions in PAFMPB for molecule AChE. (8280 atoms and 427 064 elements, MSMS meshes were used).

cores	T_{geometry}	T_{energy}	T_{rh}	T_{A}	T_{near}	T_{far}	T_{total}
1	0.36	69.74	5.24	42.87	38.01	534.30	694.02
2	0.31	36.31	2.73	22.65	19.66	277.05	362.12
4	0.28	18.47	1.41	12.26	10.18	139.28	185.43
8	0.26	9.92	0.75	13.26	5.48	72.76	105.78

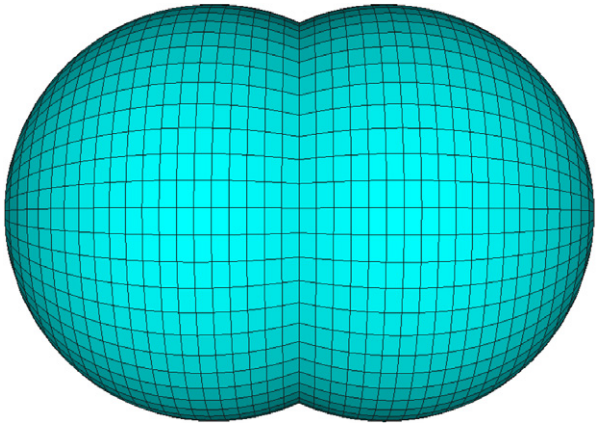

Fig. 2. Auto generated triangular (at two poles) and quadrilateral elements for a two-atom system.

geometry and mesh generation parts have poor scalability in our current code; however, their CPU time is negligible compared to other parts. We are currently studying effective ways to parallelize the geometry related parts and the matrix assembly.

4.4. Large molecular system

The previous AFMPB [6] is limited to the simulation of medium size molecules. PAFMPB uses a dynamic memory allocation technique, and efficiently simulates larger macromolecules such as the dengue virus which consists of 1082 160 atoms. In our simulation, we use the TMSmesh to obtain a mesh that has 19 502 784 elements and 9758 426 vertices. Since this mesh exceeds the memory capacity of the machine used for scaling test, the calculation is performed on an Intel Xeon X7550 machine with 2.00 GHz and 1 TB memory. PAFMPB solver terminates and output result after 70 GMRES iterations in 39 117 s using 2 cores. Fig. 3(b) shows the surface potential obtained with the PAFMPB.

4.5. Visualization

The package VCMM (visual tool for continuum molecular modeling), developed by Bai and Lu [31], can be conveniently used with PAFMPB package to visualize and analyze the surface mesh and numerical results. The output data formats for mesh and surface potential files from PAFMPB can be directly adopted by VCMM. Figs. 2 and 3 are produced by VCMM. Alternatively, VMD [32] can be applied for visualizing the simulation results [6].

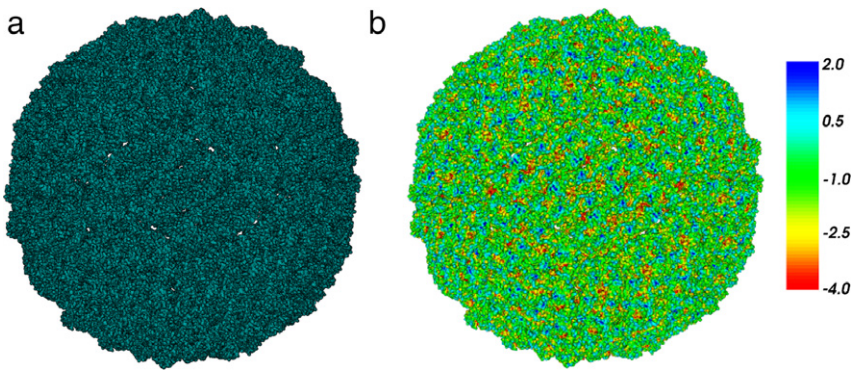

Fig. 3. Visualization of a dengue virus system using VCMM [31]. (a) A surface mesh computed by TMSmesh consists of 19 502 784 elements and 9758 426 vertices. (b) Surface potentials. Color bar is in unit of kcal/mol.e_c.

5. Conclusions

We have presented the PAFMPB solver for computing the electrostatic properties and solvation energies of biomolecular systems. Compared with the original AFMPB solver, PAFMPB utilizes the Cilk Plus runtime for parallelization, and solves the well-conditioned boundary integral equation reformulation of the linearized PB equation. Cilk Plus is easy to use with only three new keywords. Its scheduler provides a proven performance guarantee. We also integrate an automatic mesh generation option to generate the molecular VDW/SAS in the updated version; and provide a model to compute the total solvation free energy that includes both the polar and nonpolar parts. The PAFMPB package has been applied to study a virus molecule with millions of degrees of freedom on a workstation, which is impossible using our previous version.

Acknowledgments

The authors thank other members in Professor Lu's group for useful discussions and technical supports on mesh generation and 3D visualization. The work of Lu and Peng was supported by the State Key Laboratory of Scientific/Engineering Computing, National Center for Mathematics and Interdisciplinary Sciences of the Chinese Academy of Sciences, the China NSF (91230106) and 863 program (2012AA020403). The work of Zhang and Huang was supported by the National Science Foundation Awards #0905473 and #1217080, and the work of Sun and Pitsianis was supported by the National Science Foundation Award #0905164.

References

[1] N.A. Baker, D. Sept, S. Joseph, M.J. Holst, Electrostatics of nanosystems: Application to microtubules and the ribosome, Proc. Natl. Acad. Sci. USA 98 (2001) 10037–10041.
[2] J.R. Phillips, J.K. White, A precorrected-FFT method for electrostatic analysis of complicated 3-D structures, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 16 (1997) 1059–1072.
[3] W. Geng, R. Krasny, A treecode-accelerated boundary integral Poisson–Boltzmann solver for electrostatics of solvated biomolecules, J. Comput. Phys. 247 (2013) 62–78.
[4] L. Greengard, J. Huang, A new version of the fast multipole method for screened Coulomb interactions in three dimensions, J. Comput. Phys. 180 (2002) 642–658.
[5] B. Lu, X. Cheng, J. Huang, J.A. McCammon, Order N algorithm for computation of electrostatic interactions in biomolecular systems, Proc. Natl. Acad. Sci. USA 103 (2006) 19314–19319.
[6] B. Lu, X. Cheng, J. Huang, J.A. McCammon, AFMPB: An adaptive fast multipole Poisson–Boltzmann solver for calculating electrostatics in biomolecular systems, Comput. Phys. Comm. 181 (2010) 1150–1160.
[7] A.H. Boschitsch, M.O. Fenley, H. Zhou, Fast boundary element method for the linear Poisson–Boltzmann equation, J. Phys. Chem. 106 (2002) 2741–2754.
[8] 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. Commun. 182 (2011) 1272–1283.
[9] W. Geng, F. Jacob, A GPU-accelerated direct-sum boundary integral Poisson–Boltzmann solver, Comput. Phys. Commun. 184 (2013) 1490–1496.
[10] W. Geng, Parallel higher-order boundary integral electrostatics computation on molecular surfaces with curved triangulation, J. Comput. Phys. 241 (2013) 253–265.
[11] C. Li, L. Li, J. Zhang, E. Alexov, Highly efficient and exact method for parallelization of grid-based algorithms and its implementation in DelPhi, J. Comput. Chem. 33 (2012) 1960–1966.
[12] C. Li, M. Petukh, L. Li, E. Alexov, Continuous development of schemes for parallel computing of the electrostatics in biological systems: implementation in DelPhi, J. Comput. Chem. 34 (2013) 1949–1960.
[13] F. Matteo, E.L. Charles, H.R. Keith, The implementation of the Cilk-5 multithreaded language, In: Proceedings of the 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation, 1998.
[14] B.Z. Lu, Y.C. Zhou, M.J. Holst, J.A. McCammon, Recent progress in numerical methods for the Poisson–Boltzmann equation in biophysical applications, Commun. Comput. Phys. 3 (2008) 973–1009.
[15] M.F. Scanner, A.J. Olson, J.C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (1996) 305–320.
[16] M. Chen, B. Lu, TMSmesh: A robust method for molecular surface mesh generation using a trace technique, J. Chem. Theory Comput. 7 (2011) 203–212.
[17] www.continuummodel.org.
[18] X. Pang, H. Zhou, Poisson–Boltzmann calculations: van der Waals or molecular surface? Commun. Comput. Phys. 13 (2013) 1–12.
[19] B. Lu, J.A. McCammon, Improved boundary element methods for Poisson–Boltzmann electrostatic potential and force calculations, J. Chem. Theory Comput. 3 (2007) 1134–1142.
[20] B. Lee, F.M. Richards, The interpretation of protein structures: Estimation of static accessibility, J. Mol. Biol. 55 (1971) 379–400.
[21] http://www-users.cs.umn.edu/~saad/software/.
[22] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numer. 6 (1997) 229–269.
[23] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325–348.
[24] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (1975) 384–393.
[25] B. Zhang, J. Huang, N.P. Pitsianis, X. Sun, Dynamic prioritization for parallel traversal of irregularly structured spatio-temporal graphs, In: Proceedings of the 3rd USENIX on Hot Topics in Parallelism, 2011.
[26] J.A. Wagoner, N.A. Baker, Assessing implicit models for nonpolar mean solvation forces: The importance of dispersion and volume terms, Proc. Natl. Acad. Sci. USA 103 (2006) 8331–8336.
[27] R.A. Pierotti, A scaled particle theory of aqueous and nonaqueous solutions, Chem. Rev. 76 (1976) 717–726.
[28] A.V. Marenich, C.J. Cramer, D.G. Truhiar, Universal solvation model based on solute electron density and on a continuum model of the solvent defined by the bulk dielectric constant and atomic surface tensions, J. Phys. Chem. B 113 (2009) 6378–6396.
[29] I. Massova, P.A. Kollman, Combined molecular mechanical and continuum solvent approach (MM-PBSA/GBSA) to predict ligand binding, Perspect. Drug Discovery Des. 18 (2000) 113–135.
[30] K.A. Sharp, A. Nicholls, R.F. Fine, B. Honig, Reconciling the magnitude of the microscopic and macroscopic hydrophobic effects, Science 252 (1991) 106–109.
[31] S. Bai, B. Lu, VCMM: A visual tool for continuum molecular modeling, J. Mol. Graphics Modell. 50 (2014) 44–49.
[32] http://www.ks.uiuc.edu/Research/vmd/.