Computer Physics Communications 181 (2010) 1150–1160

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

AFMPB: An adaptive fast multipole Poisson–Boltzmann solver for calculating electrostatics in biomolecular systems ☆

Benzhuo Lu [a,*], Xiaolin Cheng [b], Jingfang Huang [c], J. Andrew McCammon [d]

[a] 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
[b] Center for Molecular Biophysics, Oak Ridge National Laboratory, Oak Ridge, TN 37831, United States
[c] Department of Mathematics, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3250, United States
[d] Department of Chemistry & Biochemistry, Center for Theoretical Biological Physics, Department of Pharmacology, Howard Hughes Medical Institute, University of California, San Diego, CA 92093, United States

ARTICLE INFO

Article history:
Received 23 December 2009
Accepted 16 February 2010
Available online 20 February 2010

Keywords:
Poisson–Boltzmann equation
Boundary integral equation
Node–patch method
Krylov subspace methods
Fast multipole methods
Diagonal translations

ABSTRACT

A Fortran program package is introduced for rapid evaluation of the electrostatic potentials and forces in biomolecular systems modeled by the linearized Poisson–Boltzmann equation. The numerical solver utilizes a well-conditioned boundary integral equation (BIE) formulation, a node–patch discretization scheme, a Krylov subspace iterative solver package with reverse communication protocols, and an adaptive new version of fast multipole method in which the exponential expansions are used to diagonalize the multipole-to-local translations. The program and its full description, as well as several closely related libraries and utility tools are available at http://lsec.cc.ac.cn/~lubz/afmpb.html and a mirror site at http://mccammon.ucsd.edu/. This paper is a brief summary of the program: the algorithms, the implementation and the usage.

Program summary

Program title: AFMPB: Adaptive fast multipole Poisson–Boltzmann solver
Catalogue identifier: AEGB_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEGB_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GPL 2.0
No. of lines in distributed program, including test data, etc.: 453 649
No. of bytes in distributed program, including test data, etc.: 8 764 754
Distribution format: tar.gz
Programming language: Fortran
Computer: Any
Operating system: Any
RAM: Depends on the size of the discretized biomolecular system
Classification: 3
External routines: Pre- and post-processing tools are required for generating the boundary elements and for visualization. Users can use MSMS (http://www.scripps.edu/~sanner/html/msms_home.html) for pre-processing, and VMD (http://www.ks.uiuc.edu/Research/vmd/) for visualization.
Sub-programs included: An iterative Krylov subspace solvers package from SPARSKIT by Yousef Saad (http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html), and the fast multipole methods subroutines from FMMSuite (http://www.fastmultipole.org/).
Nature of problem: Numerical solution of the linearized Poisson–Boltzmann equation that describes electrostatic interactions of molecular systems in ionic solutions.
Solution method: A novel node–patch scheme is used to discretize the well-conditioned boundary integral equation formulation of the linearized Poisson–Boltzmann equation. Various Krylov subspace solvers can be subsequently applied to solve the resulting linear system, with a bounded number of iterations independent of the number of discretized unknowns. The matrix–vector multiplication at each iteration

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

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

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2010.02.015

is accelerated by the adaptive new versions of fast multipole methods. The AFMPB solver requires other stand-alone pre-processing tools for boundary mesh generation, post-processing tools for data analysis and visualization, and can be conveniently coupled with different time stepping methods for dynamics simulation.

Restrictions: Only three or six significant digits options are provided in this version.

Unusual features: Most of the codes are in Fortran77 style. Memory allocation functions from Fortran90 and above are used in a few subroutines.

Additional comments: The current version of the codes is designed and written for single core/processor desktop machines. Check http://lsec.cc.ac.cn/~lubz/afmpb.html and http://mccammon.ucsd.edu/ for updates and changes.

Running time: The running time varies with the number of discretized elements (N) in the system and their distributions. In most cases, it scales linearly as a function of N.

1. Introduction

In the past thirty years, the Poisson–Boltzmann (PB) continuum electrostatic model has been widely accepted as a tool in theoretical studies of interactions of biomolecules such as proteins and DNAs in aqueous solutions. Recent work includes the introduction of more physical boundary conditions so that PB and its linearized version become valid for a wider class of molecular systems [30].

In this paper, we describe an adaptive fast multipole Poisson–Boltzmann (AFMPB) solver for solving the linearized PB equation, and thus for elucidating the electrostatic role in many biological processes, such as enzymatic catalysis, molecular recognition and bio-regulation. Indeed, quite a few PB solvers are available in biochemistry and biophysical communities, including DelPhi, GRASP, MEAD, UHBD and PBEQ based on the finite difference formulation [10], and APBS (adaptive Poisson–Boltzmann solver) based on the finite volume/multigrid framework [1,11,19]. In the linearized PB regime, algorithms using the boundary integral equation (BIE) approach have shown great promise for their efficiency on scaling and memory requirements [13,22,24]: when Green's theorem and potential theory are applied, the linearized PB equation can be recasted into a set of boundary integral equations where the unknowns are only defined on the surface of the molecule. Therefore, the number of unknowns is reduced when compared with the volumetric discretization in finite difference and finite element methods.

This AFMPB solver reflects our effort on developing more efficient codes using the BIE approach for the linearized PB equation, currently on single processor computing architectures. Several techniques are used in AFMPB to improve its efficiency over existing BIE-based solvers. First, a well-conditioned BIE formulation is used so that the number of iterations in the Krylov subspace methods is bounded, independent of the number of unknowns in the system. Second, a node–patch scheme is applied to discretize the resulting BIE, and the node based scheme reduces the number of unknowns defined on the molecular surface compared with commonly used "constant element" discretizations. Further, the iterative Krylov subspace methods from the SPARSKIT package [7] are applied to the resulting linear systems with simplified calling interface with the use of the reverse communication protocols. Fourth, the adaptive new versions of the fast multipole methods (FMMs) from FMMSuite [4] are applied to the convolution type discretized integrals. Finally, interface computer programs are provided to couple AFMPB with many existing mesh generating (e.g. *MSMS* [6] or other programs that can generate *OFF* format type of mesh) and visualization tools (e.g. VMD [5]). Preliminary numerical experiments show that the new AFMPB solver achieves significant speedup and memory savings for calculating the electrostatics of large-scale biomolecular systems on single-processor personal computers. We are currently parallelizing the codes on multi-core/multi-processor computer architectures toward simulating the dynamics of complex biomolecular systems under the influence of electrostatic forces derived from the PB calculation.

This paper is organized as follows. In Section 2, we describe various numerical techniques used in the AFMPB solver, including the boundary integral equation formulation for the linearized PB equation, the node–patch discretization, the Krylov subspace methods, and the adaptive new version of FMM. In Section 3, the overall structure of the codes is discussed, in particular, how it can be coupled with existing tools for pre- and post-processing. In Section 4, test runs are described to illustrate the performance of the solver. Finally, in Appendices A–C, we briefly describe the units and several important parameters used in our codes, and provide a sample shell script file for running the package.

2. Theoretical background

2.1. Electrostatics in biomolecular systems

The electrostatic force is considered an important factor in understanding the interactions and dynamics of molecular systems in solution. One commonly used continuum model for describing the electrostatic effects of the solvent outside the molecules is the Poisson–Boltzmann (PB) equation

$$-\nabla\cdot(\epsilon\nabla\phi)+k^2\sinh(\phi)=\sum_{i=1}^{M}q_i\delta(r-r_i), \tag{1}$$

and the Poisson equation is used for the inside of the molecules. When electrostatic potentials are small, the linearized PB (LPB) equation

$$-\nabla\cdot(\epsilon\nabla\phi)+k^2\phi=\sum_{i=1}^{M}q_i\delta(r-r_i) \tag{2}$$

is valid. The LPB equation can be generalized to a wider class of problems when proper interface boundary conditions are given [30].

In numerical simulations of large molecular systems, efficient solution of the Poisson–Boltzmann equation is required, for direct visualization of electrostatic potentials, for calculating the energetics of molecular interactions in solution, and more importantly, for understanding the dynamics of molecular systems in which the PBE is coupled with time evolution equations including the continuum diffusion equations and the Newton's law of motion.

2.2. Boundary integral equation formulations

There are several ways to reformulate the LPB equation as "boundary integral equations". When Green's second identity is applied directly, the traditional boundary integral equations for the linearized PB equation for a single domain (molecule) takes the form

$$\begin{aligned}
\frac{1}{2}\phi_p^{int} &= \oint_S \left[G_{pt}\frac{\partial \phi_t^{int}}{\partial n} - \frac{\partial G_{pt}}{\partial n}\phi_t^{int} \right] dS_t + \frac{1}{D_{int}}\sum_k q_k G_{pk}, \quad p \in S, \\
\frac{1}{2}\phi_p^{ext} &= \oint_S \left[-u_{pt}\frac{\partial \phi_t^{ext}}{\partial n} + \frac{\partial u_{pt}}{\partial n}\phi_t^{ext} \right] dS_t, \quad p \in S,
\end{aligned} \tag{3}$$

with the interface conditions $\phi^{int} = \phi^{ext}$ and $D_{int}\frac{\partial \phi^{int}}{\partial n} = D_{ext}\frac{\partial \phi^{ext}}{\partial n}$. In the formulas, ϕ_p^{int} is the interior potential at surface position p of the molecular domain Ω, $S = \partial\Omega$ is its boundary, i.e., solvent-accessible surface, ϕ_p^{ext} is the exterior potential at position p, D_{int} is the interior dielectric constant, t is an arbitrary point on the boundary, n is the outward normal vector at t, $\oint$ represents the principal value integral to avoid the singular point when $t \to p$ in the integral equations, $G_{pt} = \frac{1}{4\pi|r_t - r_p|}$ and $u_{pt} = \frac{\exp(-\kappa|r_t - r_p|)}{4\pi|r_t - r_p|}$ are the fundamental solutions of the corresponding Poisson and linearized Poisson–Boltzmann equations, respectively, r_k is the position of the kth source point charge q_k of the molecule, and κ is the reciprocal of the Debye–Hückel screening length determined by the ionic strength of the solution. Unfortunately, this "natural" boundary integral equation formulation is in general a Fredholm integral equation of the first kind and hence ill-conditioned. When solved iteratively using the Krylov subspace methods, the number of iterations increases with the number of unknowns. The resulting algorithm becomes inefficient for large systems.

In our AFMPB solver, instead of simply accepting this natural BIE formulation, we use the technique in [27] and combine the single and double layer potentials to derive a second kind Fredholm integral equation as in (with notations $f = \phi^{ext}$, $h = \frac{\partial \phi^{ext}}{\partial n}$)

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

where n_0 is the unit normal vector at point $p \in S$. This is a well-conditioned (Fredholm second kind) system of equations. When Krylov subspace methods are applied to such systems, the number of iterations remains bounded even for a large number of unknowns. Similar formulations are also used in [21] where a complete boundary integral equation formulation is derived for the linearized PB equation using a limiting process to avoid the singularity problem, and in later boundary element method (BEM) based PB work [13,23].

2.3. "Node–patch" discretization

To discretize Eqs. (4), one commonly used technique in engineering community is the "constant element" approach that triangularizes the molecular surface and treats the unknowns as constants on each triangular element. In this approach, the number of unknowns equals to the number of elements. Alternatively, the "linear element" method linearly interpolates the solution using the three constituent nodes, therefore the number of unknowns equals to the number of nodes. As there are less number of nodes than surface elements, the linear element approach leads to a reduction of the total number of unknowns by approximately a factor of 2. However, the node-based linear element approach introduces additional complexity in its implementation. In the AFMPB solver, we develop a "node–patch" approach to take advantage of the easy implementation of the constant element method and the reduced number of unknowns in the linear element method.

The idea of the node–patch approach is to construct a "working" patch around each node instead of directly using the facet patch (element), and assume that the unknowns are constants on each new "node–patch". These new patches are illustrated in Fig. 1, in which a "node–patch" is constructed around the ith node that has five neighboring elements. The new patch is defined by the area encircled by a set of points $\{O_1, C_1, O_2, C_2, \ldots, O_5, C_5, O_1\}$, where $\{O_l,\ l = 1, \ldots, 5\}$ are the centroids of the five adjacent triangles, and $\{C_l,\ l = 1, \ldots, 5\}$ are the midpoints of the five joint edges. Clearly, each triangular element contributes one-third of its area to the new node–patch. Therefore, for far-field integration using fast multipole method the charge on the patch can be approximated by the unknown (f or h) at the node multiplied by the total area of the node–patch (see Ref. [25] for more details), while for near-field a normal quadrature method is used as in the constant or linear element method.

There are several advantages of this "node–patch" approach. First, because of the reduction of the total number of unknowns when compared to the constant element method, the computational cost of solving the resulting linear system is accordingly reduced. The "node–patch" method does involve an additional preprocessing step for deriving the average charge on each patch. This, however, only constitutes a negligible portion of the total computation time, and the processed geometric coefficients can also be saved for repeated use in iterative solving procedures. Second, relative to the linear element method, the "node–patch" method is significantly more efficient in searching and indexing the local list when used with practical matrix storage formats such as the Harwell–Boeing sparse matrix format or modified sparse column (row) format. Finally, in the "node–patch" method, as in the constant element method, the source and target are the same – the nodes, making the application of FMM from the FMMSuite package straightforward.

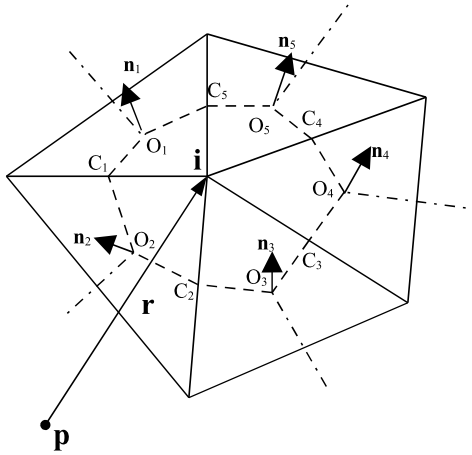

Fig. 1. "Node" patch constructed on a triangular mesh. O and n are the centroid and normal vector of an element respectively, and C is the middle point of an edge.

2.4. Krylov subspace methods

Given an initial iterate x_0, the Krylov subspace method solves the linear system $Ax = b$ by iteratively minimizing some measure of error over the space $x_0 + K_k$, where $K_k = span\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$ and r_0 is the initial residual usually defined as $r_0 = b - Ax_0$. Based on different measures of the error and different types of matrices, there are many different Krylov subspace method implementations. As the matrix A after the node–patch discretization is non-symmetric and there is no fast algorithm to apply the transpose of A to an arbitrary vector, we adopt four different Krylov iterative subroutines from the open source package SPARSKIT developed by Saad and collaborators [7,28], including the full GMRES, the restarted GMRES, the biconjugate gradients stabilized (BiCGStab) method, and the transpose free QMR (TFQMR). Since Eq. (4) is a well-conditioned Fredholm second kind integral equation system and the corresponding operator consists of an identity operator plus a compact operator whose eigenvalues only cluster at 0, the number of iterations in the Krylov subspace methods is therefore bounded, independent of the number of nodes in the discretization. Preliminary numerical experiments show that the full GMRES method converges faster than the other methods, which agrees with existing analysis. Because the memory required by the GMRES method increases linearly with the iteration number k, and the number of multiplications scales like $\frac{1}{2}k^2N$, for large k, the GMRES procedure becomes very expensive and requires excessive memory storage. For these reasons, instead of a full orthogonalization procedure, GMRES can be restarted every k_0 steps where $k_0 < N$ is some fixed integer parameter. The restarted version is often denoted as GMRES(k_0). The storages required by the BiCGStab and TFQMR algorithms are independent of the iteration number k, and the number of multiplications grows only linearly as a function of k. The choice of Krylov iterative solvers is problem dependent, and GMRES is chosen as the default solver in AFMPB.

An interesting feature of the iterative solvers in SPARSKIT is the so-called "reverse communication protocol" (check the ITSOL directory in SPARSKIT [7]), which avoids calling the matrix–vector product subroutine from inside the Krylov solver. Instead, the Krylov solver provides a vector and asks for the matrix–vector multiplication result as a further input. Therefore, it is unnecessary to pass the parameters in the FMM subroutines to the Krylov solvers, leading to easier interface between FMM and SPARSKIT, and easier memory management.

2.5. Adaptive fast multipole methods

As the total number of operations required to solve Eq. (4) is a constant (representing the number of iterations) times the amount of work required for a matrix–vector multiplication, when the new versions of fast multipole methods (FMMs) are applied, the matrix–vector product only requires $O(N)$ operations, therefore the linear equation system can be solved in asymptotically optimal $O(N)$ time.

The fundamental observation in the multipole expansion based methods is that the numerical rank of the far field interactions is relatively low and hence can be approximated by P terms (depending on the prescribed accuracy) of the so-called "multipole expansion". For the Coulomb interactions (Poisson equation),

$$\Phi(R,\theta,\phi) = \sum_{i=1}^{N} q_i \cdot \frac{1}{|\vec{R} - \vec{\rho}_i|} \approx \sum_{n=0}^{P} \sum_{m=-n}^{m=n} M_n^m \frac{Y_n^m(\theta,\phi)}{R^{n+1}} \tag{5}$$

where the multipole coefficients are computed by

$$M_n^m = 8 \sum_{i=1}^{N} q_i \cdot Y_n^{-m}(\alpha_i, \beta_i). \tag{6}$$

The spherical harmonic function of order n and degree m is defined according to the formula [8]

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(2n+1)(n-|m|)!}{4\pi(n+|m|)!}} \cdot P_n^{|m|}(\cos\theta) e^{im\phi}, \tag{7}$$

and P_n^m is the associated Legendre polynomial. For the Debye–Hückel (screened Coulombic) interaction, a similar expansion can be written as

$$\Phi(R,\theta,\phi)=\sum_{i=1}^{N} q_i\cdot\frac{e^{-\kappa|\vec{R}-\vec{\rho}_i|}}{|\vec{R}-\vec{\rho}_i|}\approx\sum_{n=0}^{P}\sum_{m=-n}^{m=n} M_n^m\cdot k_n(\kappa R)\cdot Y_n^m(\theta,\phi). \tag{8}$$

The multipole coefficients are given by

$$M_n^m=8\kappa\sum_{i=1}^{N} q_i\cdot i_n(\kappa\rho_i)\cdot Y_n^{-m}(\alpha_i,\beta_i), \tag{9}$$

where $i_n(r)$ and $k_n(r)$ are the modified spherical Bessel and modified spherical Hankel functions, respectively, defined in terms of the conventional Bessel function via [8] $I_v(r)=i^{-v}J_v(ir)$, $K_v(r)=\frac{\pi}{2\sin v\pi}[I_{-v}(r)-I_v(r)]$, $i_n(r)=\sqrt{\frac{\pi}{2r}}I_{n+1/2}(r)$, and $k_n(r)=\sqrt{\frac{\pi}{2r}}K_{n+1/2}(r)$.

For arbitrary distributions of particles, a hierarchical oct-tree (in 3D) is generated so each particle is associated with boxes at different levels, and a divide-and-conquer strategy is applied to account for the far field interactions at each level in the tree structure. In the "tree code" developed by Appel [9], and Barnes and Hut [12], as each particle interacts with 189 boxes in its "interaction list" through P terms of multipole expansions at each level and there are $O(\log N)$ levels, the total amount of operations is approximately $189P^2N\log N$. The tree code was later improved by Greengard and Rokhlin in 1987 [17]. In their original FMM, local expansions (under a different coordinate system) were introduced to accumulate information from the multipole expansions in the interaction list,

$$\Phi(R,\theta,\phi)=\sum_{i=1}^{N} q_i\cdot\frac{1}{|\vec{R}-\vec{\rho}_i|}\approx\sum_{n=0}^{P}\sum_{m=-n}^{m=n} L_n^m\cdot R^n Y_n^m(\theta,\phi) \tag{10}$$

where L_n^m are the local expansion coefficients, and for the screened Coulombic interaction, a similar expansion can be written as

$$\Phi(R,\theta,\phi)=\sum_{i=1}^{N} q_i\cdot\frac{e^{-\kappa|\vec{R}-\vec{\rho}_i|}}{|\vec{R}-\vec{\rho}_i|}\approx\sum_{n=0}^{P}\sum_{m=-n}^{m=n} L_n^m\cdot i_n(\kappa R)\cdot Y_n^m(\theta,\phi). \tag{11}$$

As the particles only interact with boxes and other particles at the finest level, and information at higher levels is transferred using a combination of multipole and local expansions, the original FMM is asymptotically optimal $O(N)$. However, because the multipole-to-local translation requires prohibitive $189P^4$ operations for each box, the huge prefactor makes the original FMM less competitive with the tree code and other FFT based methods. In 1997, a new version of FMM was introduced by Greengard and Rokhlin [18] for the Laplace equation, in which exponential expansions are introduced to diagonalize the multipole-to-local translations, and a merge-and-shift technique is used to further reduce the number of such translations. Compared with the original FMM, the original $189P^4$ operations were reduced to $40P^2+6P^3$ in the new version. Numerical experiments show that the new version of FMM breaks even with direct calculation when the number of particles $n=500$ for three digit accuracy, and $n=1000$ for six digits for the screened Coulomb interactions. In our AFMPB solver, the new version of FMM was implemented for the Laplace and linearized PB equations for the efficient calculation of electrostatic interactions. As far as we know, this is the only LPB solver using the new versions of FMMs.

Instead of discussing technical details of the adaptive new version of FMM, we refer the readers to existing literature. The new version of FMM was introduced in [18] for the Laplace equation; the corresponding adaptive version was discussed in [14]; the new version of FMM for the linearized PB equation (also called Yukawa or modified Helmholtz equation) was discussed in [16,20].

2.6. Force and torque calculations

For an interacting molecular system, AFMPB can also calculate the total force and torque acting on each molecule. The force is obtained through computing the stress tensor in ionic solution [15]

$$T_{ij}=D_{ext}E_iE_j-\frac{1}{2}D_{ext}E^2\delta_{ij}-\frac{1}{2}D_{ext}\kappa^2\phi^2\delta_{ij}, \tag{12}$$

where E is the electrostatic field, δ_{ij} is the Kronecker delta function.

To obtain the boundary stress tensor, the gradient of the potential (the negative of E on the boundary) should be computed. In our current implementation, we first use an interpolation method to construct a continuous potential function $f(x,y,z)$ in the vicinity of the molecular surface based on the PB solutions, then compute the gradient of f for any positions on the molecular surface. For more details on the interpolation method and the force calculations, we refer interested readers to [25].

Finally, the PB force F and torque M acting on any constituent molecule are calculated by integrations

$$F=\int_S T(x)\cdot dS(x), \tag{13}$$

$$M=\int_S r_c(x)\times\big[T(x)\cdot dS(x)\big], \tag{14}$$

where $r_c(x)$ is a vector from the center of mass of the target molecule to the surface point x, and the dot and cross vector multiplications are applied to the vector and tensor quantities.

3. Code structure and implementation

In this section, we describe the package portability and installation, flow chart, job running, file format and I/O layers, pre- and post-processing, and several related tools.

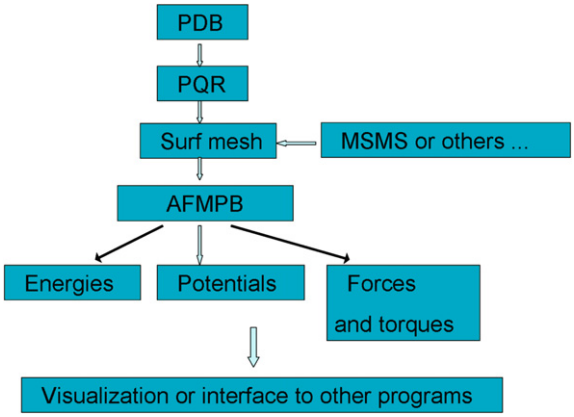

Fig. 2. AFMPB flow chart.

3.1. Portability and installation

The software package was implemented mostly in Fortran77. Memory allocation features from Fortran90 and above are used in a few subroutines for memory management.

After the user downloads the package and extracts it to a local computer, the following directories can be found:

- Doc: contains the references and this manual.
- Job-examples: contains a README file, samples of the c-shell job files *job.csh* to run the package, and the generated input/output files. A subdirectory ./Two-proteins contains similar input/output files but is for the test case of a two-molecule system.
- FBEM: the driver and the boundary integral equation codes of the AFMPB package. It also contains the subroutine iters.f, the Krylov subspace iterative solvers from SPARSKIT [7]. The default Krylov solver is GMRES, though the user can also use the restarted GMRES, TFQMR, or BiCGStab.
- fmm3d_node: the fast multipole method library for the Laplace and Yukawa potentials.
- Tools: pre- and post-processing utility programs. Additional tools will be added to this directory for calculation of different parameters.

For compiling, check the file **Makefile** for further information. The package has been successfully compiled using the Intel©compiler for Linux, and the GNU©F95 compiler.

3.2. Flow chart

The program diagram is shown in Fig. 2.

AFMPB can be considered as a main driver for calculating the electrostatics in biomolecular systems. It requires molecular structure, charge and mesh information from the pre-processing tools as inputs, and outputs potential, various energies and force results for visualization and/or further processing. To generate the "boundary elements", AFMPB currently accepts mesh data from the package *MSMS* [6] and other programs [2] that can generate molecular surface mesh in the *OFF* format. For post-processing, we use VMD for visualization. The solver can also be coupled with multi-scale time stepping schemes to simulate the dynamics of molecular systems under the influence of electrostatic interactions.

3.3. Job running

A command line style is adopted to run AFMPB. A simple way to run AFMPB is "*./afmpb*", with all the default input/output files in the working directory. A typical job with specified options can be like

afmpb -i inp.dat -o out.log -s surfpot.dat.

-i means to be followed by the input file, *-o* an output log file, and *-s* is followed by an output file that contains surface nodal potential. More options see Appendix C.

3.4. File formats and I/O layer

To run AFMPB, users are required to prepare input files including (a) a control file (*inp.dat* by default), (b) *PQR* file(s) containing atomic charges and their locations, (c) molecular surface mesh file(s), and an optional (d) off-surface points location file where the potentials need to be evaluated. For the output, a log file (*out.log* by default) will be generated automatically after every AFMPB run, and users may also request a surface potential file (default *surfpot.dat*) by setting a write-control parameter in *inp.dat*. All the input files and output log file (*out.log*) are in free format. The job-control input file can also be generated by the job shell script as described in Appendix C, in which a brief explanation for each variable is provided.

The PQR format is a modification of the PDB format which adds charge and radius parameters to existing PDB file. The surface mesh file contains node coordinates and how they connect to form triangles. The current version of AFMPB allows a few slightly different mesh formats, including the standard *OFF* format and the so-called *MSMS* format. A comment line in *inp.dat*

"# meshfmt: 0 icosohedron, 1 raw, 2 msms, 3 matlab sphere, 4 mc, 5 off"

indicates how to choose one of the optional mesh formats to be read. The *OFF* format is used by many mesh generating tools, and can be viewed using visualization software such as "GEOMVIEW", while the *MSMS* format can be conveniently generated by using a script tool provided in our package (see Section 3.7). The script will call the software MSMS [6] that is widely used in biomolecular studies. A sample *MSMS* format file is as follows:

```
5358 10712
-18.162 1.138 29.523 -0.309 -0.779 0.546
-17.185 -0.033 30.761 -0.960 0.001 -0.280
...
16.334 2.571 23.760 -0.000 1.000 0.000
1 5 3645
3645 5 3646
...
3614 3620 3613
```

In this file, the first line describes the total numbers of nodes and triangular elements. Starting from the second line, the *xyz* coordinates and optional *xyz* components of the normal direction for each node are given, followed by a list indicating the indices of three nodes for each triangular element.

The AFMPB solver requires that the normal vectors point outward (as defined in the current *OFF* and *MSMS* formats) and the nodes are ordered counterclockwisely. In case the normal vectors are not available in the input surface mesh files, which happens when other file formats are used as input files, functions are provided in the solver to calculate these quantities.

The output log file *out.log* records useful information during code execution, including the number of iterations of the iterative Krylov solver, the CPU time and memory usage information, the total/solvation/interaction/Coulombic energies, and so on. The surface potentials and forces are recorded in a formatted file (*surfpot.dat*), which can be extracted for further analysis, and can be visualized with VMD using a provided TCL script file (see Section 3.7).

3.5. Pre-processing

The *PQR* file can be generated from a standard *PDB* file using the software *pdb2pqr* [3]. *pdb2pqr* also adds missing hydrogen atoms in the *PDB* file, optimizes the hydrogen bond network, and assigns atomic charges and radii based on various force field parameter sets.

The molecular surface mesh file is traditionally generated from the *PQR* file. There exist at least three types of "molecular surface" to define the boundary between the low (interior) and high dielectric (exterior) regions: the *van der Waals* surface, the *solvent-accessible surface* [26], and the *solvent-excluded surface*. The *van der Waals* surface is the union of the surface area formed by placing van der Waals spheres at the center of each atom in a molecule. When rolling a solvent (or a probe) sphere over the van der Waals surface, the trajectory of the solvent sphere center defines the *solvent-accessible surface*, while the trajectory of the boundary of the solvent sphere in closest contact with the van der Waals surface defines the *solvent-excluded surface*. The solvent-excluded surface is also referred to as the molecular surface in bimolecular studies. In the AFMPB solver, a triangular mesh of the molecular surface can be generated using the software *MSMS* as described in [29]. Two important parameters in *MSMS* are the node density and probe radius that control the resolution of the output mesh, and the typical values are set to $1.0/\text{Å}^2$ and 1.5 Å, respectively. The mesh from *MSMS* can be further improved by removing the elements of extremely small area using a mesh checking procedure provided in the AFMPB package, though this step is usually not necessary. We want to mention that mesh generation is in general a fast step and takes only a few seconds for medium-sized molecules. A typical mesh of a molecule with 8362 atoms is shown in Fig. 3, which contains 32 975 vertices and 65 982 triangles and is generated within 3 seconds of CPU time. For larger molecules or molecular assemblies, however, our numerical experiments show that almost all existing mesh generating tools fail to generate reasonable quality molecular surface meshes required for numerical stability. The mesh generation therefore remains a bottleneck for large-scale PB calculation and full scale dynamics simulations with PB forces.

3.6. Post-processing

The calculation results for energies and forces (between molecules) can be extracted from *out.log*. The surface and off-surface potential files can be used for further analysis. The solvation energies for individual molecules are also stored in *out.log* if the corresponding output options are switched on in the input file *inp.dat*. The solvation energy is computed by solving the PBE only once, which is different from most finite difference based methods. For multi-molecule systems, the binding energy, the total forces and torques acting on individual molecules due to the other molecules (excluding itself) can also be calculated and outputed to *out.log* by setting the corresponding control variables in *inp.dat*.

3.7. Utility tools

The directory *./tools* contains tools for file format conversion, mesh generation and refinement, and data analysis and visualization. In the current release of the solver, two scripts are provided: *pqrmsms.csh* and *showSmoothMesh.tcl*. The c-shell script *pqrmsms.csh* generates a *MSMS* molecular surface mesh from a *PQR* file. Before using the script file, the program *MSMS* should be installed and the path should be correctly set. Given a *PQR* file (e.g. *fas2.pqr* in the directory), the surface mesh can be generated by running the following command:

./pqrmsms.csh fas2 1.2 1.5.

The last two variables specify the node density (in unit of 1 per Å^2) and probe radius in unit of Å, respectively.

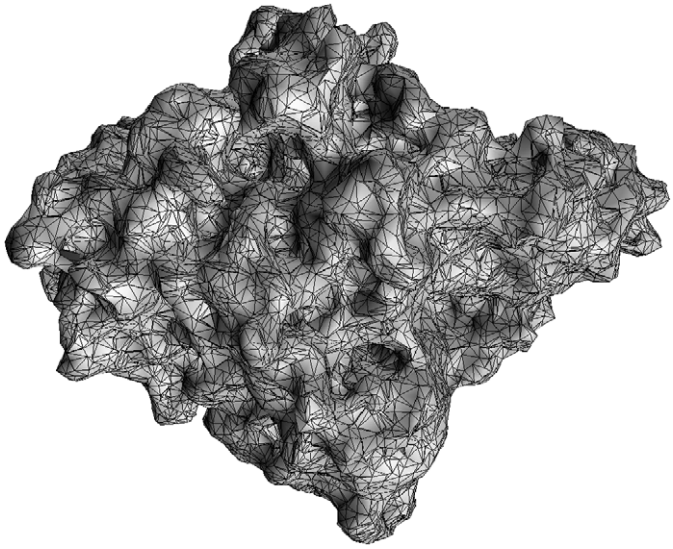

Fig. 3. A typical surface triangulated mesh of a protein (acetylcholinsterase).

Table 1
BEM performance on a spherical cavity case with different surface mesh sizes.

Number of elements	$T_{direct\ BEM}$ (s)	T_{AFMPB} (s)	Level	Iteration steps	$E_{solvation}$ (error %)	Error (%) in	
						f	h
320	0.13	0.18	2	5	−4227.5 (4.5)	6.6	5.6
1280	1.56	0.82	3	5	−4134.5 (2.2)	2.8	2.5
5120	19.67	3.39	3	5	−4088.6 (1.1)	1.4	1.1
20 480	247.20	15.86	4	5	−4066.5 (0.5)	0.7	0.6
81 920	3122.10	87.96	5	5	−4050.6 (0.3)	0.2	0.4

The second tool, a TCL script *showSmoothMesh.tcl*, is used together with the visualization program *VMD* to display the surface potential data file. *VMD* should be installed before running the script. A sample execution to visualize the *fas2* structure and the resulting surface potentials follows:

vmd fas2.pqr -e showSmoothMesh.tcl -args ../job-examples/surfpot.dat.

4. Test run description

4.1. Spherical cavity

To assess the performance of the AFMPB solver, we first calculate the Born solvation energy of a point charge +50e located at the center of a spherical cavity with a radius of 50 Å. The exact Born solvation energy $E_{solvation}$ of the cavity is −4046.0 (energy is in kcal/mol). The surface is discretized at various resolution levels by recursively subdividing an icosahedron. Table 1 summarizes the timing results (on a Dell dual 2.0 GHz P4 desktop with 2 GB memory) along with some critical control parameters using the AFMPB solver and a direct BEM without invoking any fast algorithms (denoted by direct BEM). We noticed that regardless of the surface resolution, all the GMRES iteration steps are below 5, which numerically confirms that our formulation is well-conditioned. The CPU time for the new version of FMM increases linearly with the number of elements, while increases quadratically for the direct BEM method.

The computational accuracy of the solver improves linearly upon the refinement of the mesh scale. As shown in Table 1, when the mesh is refined to a higher level, the number of elements is quadrupled, and the size of each element (for instance, measured by the length of edges of the triangular elements) is reduced by half, the relative errors of the calculated Born solvation energy are also reduced by approximately half when compared with the analytical result.

4.2. Fasciculin

To further illustrate the AFMPB's performance on the electrostatic calculation for proteins, we computed the electrostatic solvation energies of fasciculinII, a 68 residue protein at various mesh resolutions. The results are shown in Table 2. It shows again that the mesh size has only a small impact on the number of iteration steps, and the memory usage and CPU time increase almost linearly with the increase of surface meshes.

4.3. A two-molecule case: Fasciculin–acetylcholinesterase system

AFMPB is also tested for a two-molecule system, the acetylcholinesterase and its inhibitor fasciculin. They are separated by 42.2 Å away by shifting the coordinates along the center-to-center direction in the original structure. The molecular surface mesh for fasciculin contains 5198 nodes and 10 400 triangles, and the mesh for acetylcholinesterase contains 34 111 nodes and 68 266 triangles. In order to

Table 2
Performance for calculations on a single protein.

Mesh	Memory (MB)	CPU (s)	$E_{solvation}$ (kcal/mol)	Iteration steps
9920 F, 4962 V	73.9	6.7	−556.3	11
12 150 F, 6077 V	86.1	6.8	−542.5	9
15 146 F, 7575 V	118.3	8.0	−536.9	9
18 138 F, 9071 V	144.0	9.2	−527.0	9
21 684 F, 10 844 V	177.0	12.5	−524.6	10

F denotes the number of faces, V the vertices.

Table 3
Units in AFMPB.

Variable	AFMPB	SI
Length	1 Å	1×10^{-10} m
Energy	1 kcal/mol	4186 J/mol
Charge	1 electron	1.602×10^{-19} C
Force	1 kcal/(mol Å)	6.948×10^{-11} N
Ionic strength	1 mM	1×10^{-3} M
Temperature	1 K (kelvin)	1 K (kelvin)
Permittivity*	1 ϵ_0	8.854×10^{-12} F/m

* The relative permittivity is used in input file. ϵ_0 is the permittivity in vacuum.

extract various quantities characterizing the interaction between the two molecules, AFMPB actually solves the PBE three times: two for each isolated molecule, and one for the combined two-molecule system. The total solvation energy of the system is −3663.8 kcal/mol. The electrostatic interaction energy between fasciculin and acetylcholinesterase is 0.30 kcal/mol, and the total force (in kcal/(mol Å)) and torque (in kcal/mol) acting on fasciculin are $(0.08, 0.14, -0.32)$ and $(12.53, 9.95, -7.42)$, respectively. The total CPU time is 73 seconds on a DELL PRECISION T7400 desktop computer.

5. Conclusion and acknowledgements

In this paper, we describe an adaptive fast multipole Poisson–Boltzmann solver for computing the electrostatics in biomolecules. The solver uses the new version of fast multipole methods and Krylov subspace methods to solve a well-conditioned boundary integral equation formulation of the linearized PB equation. Numerical experiments show that the solver is very efficient for relatively large molecules on current desktop computers. We anticipate that the AFMPB solver, when parallelized on multi-core/multi-processor large-scale computers, can be applied to the efficient simulation of the dynamics of large molecular systems.

Finally in this paper, we want to thank many of our colleagues and collaborators for their contributions and suggestions. In particular, our code uses many existing open source codes, including the SPARSKIT by Saad and collaborators [7], *MSMS* [6] for mesh generation, VMD [5] for visualization, and several important subroutines in the new version of FMM from Profs. Greengard and Rokhlin's groups. This work was supported by HHMI, NIH, NBCR, NSF (J.H.: NSF0811130 and NSF0411920, J.A.M.: MCB0506593), the NSF Center of Theoretical Biological Physics (CTBP), and the Institute for Mathematics and Its Applications (IMA). In addition, B.L. is partially funded by the Academy of Mathematics and Systems Science of Chinese Academy of Sciences, the State Key Laboratory of Scientific/Engineering Computing, and NSFC (NSFC10971218), and X.C. is partially supported by the U.S. Department of Energy Field Work Proposal ERKJE84. Their support is thankfully acknowledged.

Appendix A. Units

Table 3 lists the units used in AFMPB. The conversion factors to transform to SI units are also given in the last column for reference.

Appendix B. Important parameters

We recommend advanced users to check the following header files for adjustable parameters used in the program.

- defgeom.h: defines the variables describing the geometry of the molecules.
- files.h: defines the units for various input and output files.
- fmmtree.h: in the fast multipole method, the size of many variables and vectors can be determined before one knows the adaptive tree structure for an input geometry. In this file, all the "fixed" length variables are defined, including many variables for storing the precomputed data. One common block is defined in this file for the adaptive tree structure.
- membem.h: defines the geometry and input/output variables in the BEM subroutines. The common block geomol defines the numbers of molecules, node points, elements, and singular charges. The common block membem defines the pointer to a huge working array where different geometry and input/output variables are stored.
- memfmm.h: defines the pointers for the variables in the FMM subroutines. All adjustable variables are allocated once the adaptive tree structure is determined, and these pointers are generated for integer, real, and complex variables, respectively.
- parm.h: defines important physical and code parameters. In particular, the following parameters are used for code control.
  - lw: as the tree structure is adaptive, one cannot determine the size of the required memory where the oct-tree structure will be stored. Hence initially a large amount of memory space is allocated to generate the adaptive tree. If this number is too small, error message will be provided asking the user to increase this number.

- nbox: in our adaptive strategy, we ask that the maximum number of particles in each childless box is less than nbox. Reducing this parameter will generate more levels (which means more boxes but less direct interactions).
- nterms, nlambs: the number of multipole and exponential expansions in the FMM code. Currently only 3 and 6 digits accuracy are allowed, corresponding to nterms=nlambs=9 and nterms=nlambs=18, respectively.

Appendix C. A sample shell script

In the following, we provide a simple shell file for executing the AFMPB package.

A Shell file for AFMPB

```
#!/bin/csh
# xxx, 20xx
# Set up directories.
  set DIR=../
  set OUT=.

# Set up molecule data and mesh.
  set pqr1=fas2.pqr # mol1 pqr
  set mesh1=fas2-mod.dat-d1.2-r1.5 # mol1 msms mesh

# Prepare input data file.
  cat << _END > inp.dat
  # nmol, number of molecules
  1
  # di, de, ion concentration(mM), temperature
  2.0 80. 150 300.0
  # meshfmt: 0 icosohedron, 1 raw, 2 msms, 3 matlab sphere, 4 mc, 5 off
  2
  # output key: ipotw(surf pot),iforce,iinterE,iselfE (solvation),ipotdx(vol pot)
  1 0 0 1 0
  # pqr and mesh files (repeat nmol lines for multi-molecules)
  $pqr1 $mesh1
  _END

# Start the code.
  $DIR/afmpb -i inp.dat -o $OUT/out.log -s $OUT/surfpot.dat
```

AFMPB also supports potential calculations at exterior points given in *space_mesh.m* file. A sample command line for this option follows, where the computed potentials are stored in *evo.dat*.

afmpb -i inp.dat -vm ./space_mesh.m -o ./out.log -s ./pot.dat -vp ./evo.dat.

Finally, a simple command line "${PROG_PATH}/afmpb" can be used with all the default options, and the output files will be generated in the working directory.

References

[1] http://apbs.sourceforge.net/.
[2] http://fetk.org/codes/gamer/.
[3] http://pdb2pqr.sourceforge.net/.
[4] http://www.fastmultipole.org/.
[5] http://www.ks.uiuc.edu/Research/vmd/.
[6] http://www.scripps.edu/~sanner/html/msms_home.html.
[7] http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html.
[8] M. Abramowitz, I.A. Stegun, Handbook of Mathematical Functions, Dover, 1970.
[9] A.W. Appel, SIAM J. Sci. Statist. Comput. 6 (1985) 85–103.
[10] N.A. Baker, Methods in Enzymol. 383 (2004) 94–118.
[11] N.A. Baker, D. Sept, S. Joseph, M.J. Holst, J.A. McCammon, Proc. Natl. Acad. Sci. USA 98 (2001) 10037.
[12] J. Barnes, P. Hut, Nature 324 (1986) 446–449.
[13] A.H. Boschitsch, M.O. Fenley, H.X. Zhou, J. Phys. Chem. B 106 (2002) 2741–2754.
[14] H. Cheng, L. Greengard, V. Rokhlin, J. Comput. Phys. 155 (1999) 468–498.
[15] M.K. Gilson, M.E. Davis, B.A. Luty, J.A. McCammon, J. Phys. Chem. 97 (1993) 3591–3600.
[16] L. Greengard, J.F. Huang, J. Comput. Phys. 180 (2002) 642–658.
[17] L. Greengard, V. Rokhlin, J. Comput. Phys. 73 (1987) 325–348.
[18] L. Greengard, V. Rokhlin, Acta Numer. 6 (1997) 229–269.
[19] M. Holst, N.A. Baker, F. Wang, J. Comput. Chem. 21 (2000) 1319.
[20] J. Huang, B. Zhang, J. Jia, Computer Physics Communications 180 (2009) 2331–2338.
[21] A.H. Juffer, E.F.F. Botta, B.A.M. Vankeulen, A. Vanderploeg, H.J.C. Berendsen, J. Comput. Phys. 97 (1) (1991) 144–171.
[22] S. Kuo, M. Altman, J. Bardhan, B. Tidor, J. White, in: Proc. IEEE/ACM Int. Conf. Computer Aided-Design, 2002, pp. 466–473.

[23] J. Liang, S. Subramaniam, Biophys. J. 73 (4) (1997) 1830–1841.
[24] B. Lu, X. Cheng, J. Huang, J.A. McCammon, Proc. Natl. Acad. Sci. USA 103 (2006) 19314–19319.
[25] B. Lu, J.A. McCammon, J. Chem. Theory Comput. 3 (2007) 1134.
[26] F.M. Richards, Ann. Rev. Biophys. Bioengrg. 6 (1977) 151–176.
[27] V. Rokhlin, Wave Motion 5 (3) (1983) 257–272.
[28] Y. Saad, M. Schultz, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[29] M.F. Sanner, A.J. Olson, J.C. Spehner, Biopolymers 38 (1996) 305–320.
[30] C.C. Lee, H.J. Lee, Y.K. Hyon, T.C. Lin, C. Liu, personal communication.