Commun. Comput. Phys.
doi: 10.4208/cicp.210711.111111s

Vol. **13**, No. 1, pp. 107-128
January 2013

Mathematical and Numerical Aspects of the Adaptive Fast Multipole Poisson-Boltzmann Solver

Bo Zhang[1], Benzhuo Lu[2], Xiaolin Cheng[3], Jingfang Huang[4,*], Nikos P. Pitsianis[1,5], Xiaobai Sun[1] and J. Andrew McCammon[6]

[1] *Department of Computer Science, Duke University, NC 27708, USA.*
[2] *Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100910, China.*
[3] *Center for Molecular Biophysics, Oak Ridge National Laboratory, TN 37831, USA.*
[4] *Department of Mathematics, University of North Carolina, Chapel Hill, NC 27599, USA.*
[5] *Department of Electrical and Computer Engineering, Aristotle University, Thessaloniki, 54124, Greece.*
[6] *Department of Chemistry & Biochemistry, Center for Theoretical Biological Physics, Department of Pharmacology, Howard Hughes Medical Institute, University of California, San Diego, CA 92093, USA.*

Received 21 July 2011; Accepted (in revised version) 11 November 2011

Available online 12 June 2012

Abstract. This paper summarizes the mathematical and numerical theories and computational elements of the adaptive fast multipole Poisson-Boltzmann (AFMPB) solver. We introduce and discuss the following components in order: the Poisson-Boltzmann model, boundary integral equation reformulation, surface mesh generation, the node-patch discretization approach, Krylov iterative methods, the new version of fast multipole methods (FMMs), and a dynamic prioritization technique for scheduling parallel operations. For each component, we also remark on feasible approaches for further improvements in efficiency, accuracy and applicability of the AFMPB solver to large-scale long-time molecular dynamics simulations. The potential of the solver is demonstrated with preliminary numerical results.

AMS subject classifications: 45B05, 65Y05, 68W10, 90B10, 92C05, 92C40

Key words: Biomolecular system, electrostatics, Poisson-Boltzmann equation, fast multipole methods, mesh generation, directed acyclic graph, dynamic prioritization, parallelization.

*Corresponding author. *Email addresses:* zhangb@cs.duke.edu (B. Zhang), bzlu@lsec.cc.ac.cn (B. Lu), chengx@ornl.gov (X. Cheng), huang@email.unc.edu (J. Huang), Nikos.P.Pitsianis@duke.edu (N. P. Pitsianis), xiaobai@cs.duke.edu (X. Sun), jmccammo@ucsd.edu (J. A. McCammon)

1 Introduction

In the past three decades, the Poisson-Boltzmann (PB) continuum electrostatic model has been adopted in many simulation tools for theoretical studies of electrostatic interactions between biomolecules such as proteins and DNAs in aqueous solutions. Various numerical techniques have been developed to solve the PB equations and help elucidate the electrostatic role in many biological processes, such as enzymatic catalysis, molecular recognition and bioregulation. Existing simulation packages or PB solvers use the finite difference method, such as in DelPhi [1], GRASP [2, 47], MEAD [13], UHBD [3, 46], PBEQ [35, 36], PB solver [32] in AMBER [18], ZAP [27] and MIBPB [62], or use the finite volume/multigrid method, such as in the Adaptive Poisson-Boltzmann Solver (APBS) [4]. In a circumstance where the linearized PB is applicable, the partial differential equations can be reformulated into a set of surface integral equations (IEs) by using Green's theorem and potential theory. The unknowns in the IEs are located on the molecular surface only, and the resulting discretized linear system can be solved efficiently and accurately with certain fast convolution algorithms, in particular, the fast Fourier transform (FFT) and the fast multipole method (FMM).

The main purpose of this paper is to introduce the adaptive fast multipole Poisson-Boltzmann (AFMPB) solver, in the aspects of mathematical theories, numerical properties and computational components. The numerical components of the AFMPB are mostly based on previously published results, by some of the authors and other researchers. We give a brief summary and provide certain references, not exhaustively, to the precursor work. Certain computational components, especially for algorithmic parallelization and parallel scheduling, are recently developed by the authors. We introduce them briefly. We emphasize that even when each component is well studied and understood, a coherent integration of these components still calls for special attention and efforts. The success of a mathematical software is ultimately measured by its applicability and the extent of its applications. The rest of the paper is organized into two sections. In Section 2, we describe the PB model, the boundary integral equation (BIE) reformulation, the surface mesh generation, a node-patch discretization approach, the Krylov subspace methods, the new version of FMMs, and a dynamic prioritization technique for parallelization. We comment on each topic certain feasible strategies or active efforts for further improvements in efficiency and accuracy. In Section 3, we present numerical results from preliminary experiments and demonstrate applicability and performance of the AFMPB solver.

2 Theoretical foundations and computational elements

2.1 Continuum Poisson-Boltzmann electrostatics model

The electrostatic force is considered to play an important role in the interactions and dynamics of molecular systems in aqueous solution. In the Poisson equation model, when the charge density that describes the electrostatic effects of the solvent outside the

molecules is approximated by a Boltzmann distribution, the continuum nonlinear PB equation assumes the following familiar form

$$-\nabla\cdot(\epsilon\nabla\phi)+\bar{\kappa}^2\sinh(\phi)=\sum_{i=1}^{M}q_i\delta(r-r_i). \tag{2.1}$$

In the formula, the molecule is represented by M point charges q_i located at r_i, ϵ is the (position-dependent) dielectric constant, ϕ is the electrostatic potential at location r, $\bar{\kappa}$ is the *modified* Debye-Hückel parameter, $\bar{\kappa}=0$ in the molecule region and $\bar{\kappa}=\sqrt{\epsilon}\kappa$ in the solution region, κ is the inverse of the Debye-Hückel screening length determined by the ionic strength of the solution. Comprehensive introductions or reviews of the implicit solvent models and the PB theory can be found in the literature [24,48,52]. The PB theory is also known as the Gouy-Chapman (GC) theory in electrochemistry, the Debye-Hückel theory in solution chemistry, and the Derjaguin-Landau-Verwey-Overbeek (DLVO) theory in colloid chemistry, respectively. When the electrostatic potentials are small, the linearized PB (LPB) equation

$$-\nabla\cdot(\epsilon\nabla\phi)+\bar{\kappa}^2\phi=\sum_{i=1}^{M}q_i\delta(r-r_i) \tag{2.2}$$

becomes valid, equipped with the interface conditions $[\phi]=0$ (by the continuity of the potential) and $[\epsilon\frac{\partial\phi}{\partial n}]=0$ (by the conservation of flux). Here, $[\ ]$ denotes the jump across the molecular surface and $\frac{\partial}{\partial n}$ is the outward (into the solvent) normal direction gradient at the surface.

It is observed in experiments and numerical simulations that the nonlinear PB model is sufficiently accurate for highly charged molecular systems. For numerical solution of the nonlinear PB model, many existing solvers use finite difference methods by the formulation of partial differential equations, or finite element methods by certain variational formulations. However, the efficiency in solving the PB equations matters greatly for the study of large molecular systems via simulation, which involves direct visualization of the electrostatic potentials and intensive calculation of the energetics of molecular interactions in the solution. Highly efficient solutions are essential for simulating the dynamics of a molecular system for a biologically relevant time period, in which the PB equation is coupled with time evolution equations describing, for example, continuum diffusion and Newton's law of motion.

It is also noticed that the nonlinear effects are often restricted to a small neighborhood of the molecular surface. This opens up new opportunities for efficient solution of the nonlinear PB model. To our knowledge, there are ongoing efforts on improving the LPB model with adequate interface conditions in order to yield results in better agreement with that by the nonlinear PB model. In other words, such a remodeling approach attempts to capture the nonlinearity in the interface conditions instead. If successful, it will have computational advantages in that the numerical solution for the LPB model requires only the solution of constant coefficient linear equations, and it allows the model to

be reformulated into BIEs and then solved with efficient numerical methods [16,42]. The solution approaches for the nonlinear models so far resort only to volume discretization methods using finite differences or finite elements.

2.2 Boundary integral equation reformulation

With the LPB model, one can reformulate the partial differential equation into BIEs. An immediate benefit is the reduction in the number of unknown variables upon discretization. There exists a traditional way to do so, but we introduce a better BIE reformulation approach that yields well-conditioned discrete equations and leads to both efficient, stable and accurate solutions.

Traditionally, one applies Green's second identity to derive the following BIE for the LPB equation over a single domain (molecule),

$$\frac{1}{2}\phi_p^{int} = \oint_S \left[G_{pt}\frac{\partial \phi_t^{int}}{\partial n} - \frac{\partial G_{pt}}{\partial n}\phi_t^{int} \right] dS_t + \frac{1}{\epsilon_{int}}\sum_k q_k G_{pk}, \quad p \in S = \partial\Omega, \tag{2.3a}$$

$$\frac{1}{2}\phi_p^{ext} = \oint_S \left[-u_{pt}\frac{\partial \phi_t^{ext}}{\partial n} + \frac{\partial u_{pt}}{\partial n}\phi_t^{ext} \right] dS_t, \qquad p \in S, \tag{2.3b}$$

with the interface conditions

$$\phi^{int} = \phi^{ext}, \qquad \epsilon_{int}\frac{\partial \phi^{int}}{\partial n} = \epsilon_{ext}\frac{\partial \phi^{ext}}{\partial n},$$

where $S = \partial\Omega$ is the boundary of the molecule. It is assumed that S is smooth; otherwise, the BIEs should be modified for nonsmooth surfaces. The potentials at a surface position p in the interior/molecular region and exterior/solution regions are denoted by ϕ_p^{int} and ϕ_p^{ext}, respectively. In addition, ϵ_{int} and ϵ_{ext} are respectively the interior and exterior dielectric constants, t is an arbitrary point on the boundary and n is the outward normal vector at t, $\oint$ denotes the principal value integral to avoid the singular point when $t \to p$ in Eq. (2.3). The functions G_{pt} and u_{pt} are the fundamental solutions to the corresponding Poisson and LPB equations,

$$G_{pt} = \frac{1}{4\pi|r_t - r_p|}, \qquad u_{pt} = \frac{e^{-\kappa|r_t - r_p|}}{4\pi|r_t - r_p|},$$

with r_k as the position of the kth source point for charge q_k of the molecule, and κ as the reciprocal of the Debye-Hückel screening length. One notices that the unknowns are placed on the molecular surface only.

The BIE formulation (2.3), together with the interface conditions, results in a Fredholm integral equation of the first kind, which is unfortunately ill-conditioned, leading to any discrete system progressively ill-conditioned as the number of unknowns increases. The increase in the condition number typically implies the increase in the number of iterations and hence the arithmetic complexity of an iterative method, such as a Krylov

subspace method. One may use certain preconditioners if available. Such preconditioning may be viewed as a numerical reformulation technique.

Based on the work of Rokhlin [51] we developed an alternative reformulation approach for AFMPB and arrived at Fredholm equations of the second kind. Particularly, we have the following equations,

$$\left(\frac{1}{2\bar{\epsilon}}+\frac{1}{2}\right)f_p=\oint_S\left[(G_{pt}-u_{pt})h_t-\left(\frac{1}{\bar{\epsilon}}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)f_t\right]dS_t+\frac{1}{\epsilon_{ext}}\sum_k q_k G_{pk}, \tag{2.4a}$$

$$\begin{aligned}\left(\frac{1}{2\bar{\epsilon}}+\frac{1}{2}\right)h_p=&\oint_S\left[\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\bar{\epsilon}}\frac{\partial u_{pt}}{\partial n_0}\right)h_t-\frac{1}{\bar{\epsilon}}\left(\frac{\partial^2 G_{pt}}{\partial n_0\partial n}-\frac{\partial^2 u_{pt}}{\partial n_0\partial n}\right)f_t\right]dS_t\\&+\frac{1}{\epsilon_{ext}}\sum_k q_k\frac{\partial G_{pk}}{\partial n_0},\end{aligned} \tag{2.4b}$$

with

$$f=\phi^{ext},\qquad h=\frac{\partial\phi^{ext}}{\partial n},$$

where n_0 is the outward unit normal vector at point $p\in S$ and $\bar{\epsilon}=\frac{\epsilon_{ext}}{\epsilon_{int}}$. This system of equations is well conditioned. It provides the foundation for a well-conditioned system of equations with an adequate discretization method. This reformulation approach may be seen as an analytical technique for preconditioning. Our numerical results with certain Krylov subspace methods on Eq. (2.4) show that the number of iterations remains bounded up to a large number of unknowns.

Similar formulations are previously derived by others for PB solutions. Juffer et al. used a limiting process to circumvent the singularity problem [37]. Some others used the boundary element method (BEM) [17, 41]. Liang and Subramaniam provided comparisons between the first and second kind formulations [41]. We comment that the second kind Fredholm integral equation representation is not unique in general. It is an interesting research topic to find the best BIE representation for optimal efficiency and accuracy by certain criteria. Also, numerical preconditioning strategies may be combined with BIE methods in the physical or frequency domain to increase the convergence rate.

2.3 Molecular surface and mesh generation

Numerical solution to the BIE formulation in Eq. (2.4) requires a description of the molecular surface S and a high quality mesh for numerical discretization. This is a preprocessing step in the AFMPB solver. The AFMPB solver accepts mesh data from popular molecular surface calculation packages such as MSMS [5], GAMer (Geometry-preserving Adaptive MeshER) [6], TMSmesh (Tracing Molecular Surface for meshing) [7,19], or a triangulated molecular surface mesh generated by other programs. Fig. 1 shows the mesh generated by TMSmesh for an ion channel protein structure. The protein is a proton-gated ion channel from Gloeobacter violaceus (GLIC, PDB code: 3ehz), and the structure is taken from a MD simulation trajectory [25].

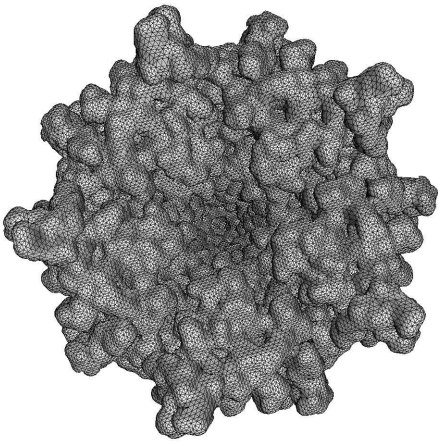

Figure 1: Surface triangular mesh of a channel protein (GLIC) structure.

There exist several definitions of molecular surfaces, such as the van der Waals (VDW) surface [15] defined as the union of the VDW surfaces of individual atoms, the solvent accessible surface (SAS) [40] and solvent excluded surface (SES) [50] formed by rolling a probe sphere on the VDW surface, the minimal energy molecular surface [14, 23] derived by minimizing the surface free energy, and the Gaussian surface [61, 64] defined as the level set of summations of the Gaussian kernel functions as adopted in TMSmesh [7]. Software packages based on these surface definitions generate acceptable meshes for visualization purposes, but lead to differences in the calculation of free energy and electrostatic properties. Also, some of them display poor performance in efficiency or numerical stability. For example, we observed that MSMS, although highly efficient, generated from time to time elongated triangular elements of extremely small area, isolated nodes or edges. A low quality mesh may significantly slow down or even inhibit the convergence of the Krylov iterative methods used in AFMPB (see also Section 3). For approximating molecular surfaces, the recently developed software TMSmesh [7, 19] usually generates better quality meshes and can handle very large-scale molecules such as viruses that challenge many other existing mesh generation tools. There are other active efforts to improve mesh quality, such as proper filtering of coarse triangular meshes, e.g., using Delaunay conforming triangulation [26].

In the efficiency aspect, based on our experiments, the sequential mesh generation tools mentioned above typically take from seconds to a few minutes each for a system with a few to a dozen thousand atoms. This is comparable to the time the sequential AFMPB solver takes. While such computation time may be sufficient for single structure calculation, in which the mesh is generated only once, we face a pressing challenge in efficiency for calculating an ensemble of structures that may undergo changes in the dynamics simulations. Such calculation requires remeshing or mesh update at each time step. A main approach to dealing with the efficiency issue is to design and develop *parallel* mesh generation tools and PB solvers; see the work from D'Agostino et al. [21].

2.4 The "node-patch" discretization method

For the AFMPB solver, we have developed a "node-patch" approach [44] to discretizing the integral equations (2.4). It involves a transformation of the initial surface mesh provided by a mesh generation tool. The initial mesh is typically composed of flat triangular facet patches. The node-patch approach makes use of two low-order BEM techniques widely used in the engineering community, namely, the piecewise constant method and the linear element method. In the constant element method, the number of unknowns equals to the number of triangular patches, and each unknown is treated as a constant on its triangular patch. In the linear element method, the number of unknowns equals to the number of (vertex) nodes, and the solution on each patch is interpolated linearly using values at the three vertex nodes. The number of nodes is approximately a half of the number of triangular elements. The linear element method leads to a reduction in the number of unknowns, but requires more sophisticated formulas and numerical integration schemes when evaluating the local direct interactions (panel integrations). Its implementation is consequently more complicated.

In the node-patch approach, we combine the constant element method for its easy implementation and the linear element method for its reduced number of unknowns. We first construct a *virtual* patch or node-patch around each node to replace the facet patch (element); see an illustration in Fig. 2 for the node-patch around the vertex i, which is the intersection set of the five triangular elements. Specifically, the patch area is enclosed by the dashed edges connecting the centroid points $\{O_1,O_2,\cdots,O_5\}$ of the triangular elements and the midpoints $\{C_1,C_2,\cdots,C_5\}$ of the edges incident at i. Next, the unknowns are assumed constant on each new node-patch. Clearly, each triangular element contributes one third of its area to the new node-patch. Thus, for the far-field integration, the charge on the patch can be approximated by the charge at the node multiplied by the total area of the node-patch; for the near-field integration, a normal quadrature method is used.

The node-patch approach has a couple of advantages. (1) The computational cost of the resulting linear system is reduced with the number of unknowns. The additional cost for deriving the average charge on each patch is negligible in comparison to the time saved. Furthermore, the geometric coefficients produced in this phase can be saved for repeated use in an iterative solution process. (2) One can split the matrix into two additive terms, by certain cutoff criteria. One of the terms is for the direct interactions with the "near points" (local list panel integrations), such as specified as the local neighbor box list in the FMM. The calculation and storage processing of the near-interaction matrix by the node-patch approach are more economic compared to that by the linear element approach. This reduces substantial execution time spent on non-numerical operations in retrieving datum entries. It is long recognized that when the arithmetic complexity is nearly linear, the computation time may be dominated by data accesses and other nonarithmetic operations.

There are other active research efforts for solutions of PB models with smooth sur-

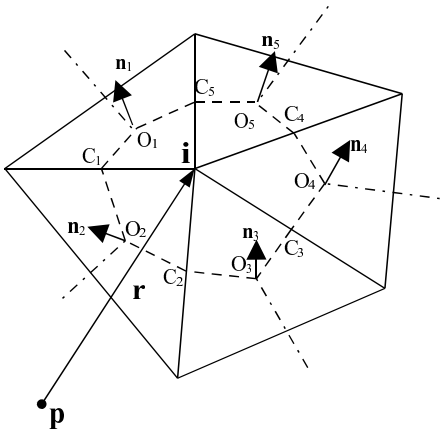

Figure 2: Illustration of a node-patch construction around node i in a triangulated mesh: O_ℓ and n_ℓ are the centroid and normal vectors of element ℓ, respectively, C_ℓ is the middle point of edge ℓ incident at node i, $\ell=1,\cdots,5$.

faces, such as on high-order surface description, or on approximation of the solution on the surface with higher accuracy [38, 39]. As mentioned earlier, with a nonsmooth surface, the interface conditions must be changed accordingly. We are aware that there exist different definitions and descriptions for a smooth *molecular* surface by domain scientists.

2.5 Krylov subspace methods

The discretized linear system from Eq. (2.4) can be expressed in matrix-vector form as $Ax=b$. AFMPB solves this system with Krylov subspace methods. Given an initial iterate x_0, a Krylov method solves the linear system by minimizing the iterative errors, with respect to a selected measure, in the so-called Krylov space x_0+K_k at step k, where $K_k=\text{span}\{r_0, Ar_0, A^2r_0, \cdots, A^{k-1}r_0\}$ with $r_0=b-Ax_0$. The basic operations in a Krylov subspace method are matrix-vector multiplications. Krylov subspace methods differ from each other in their assumptions on the structures or properties of the linear system, such as symmetric or nonsymmetric, with or without an available routine for the product of A^T and a vector if the matrix is nonsymmetric, with or without defective eigenspaces.

The matrix A formed by the node-patch discretization of Eq. (2.4) is usually nonsymmetric. There is no known or available fast algorithm for applying the transpose of A to an arbitrary vector. AFMPB employs four particular Krylov iterative subroutines for solving nonsymmetric systems of linear equations, especially provided by the open source package SPARSKIT [8]. They are the full GMRES, the restarted GMRES, the biconjugate gradient stabilized (BiCGStab) method, and the transpose-free Quasi-Minimum Residual (TFQMR) method. SPARSKIT follows the so-called "reverse communication protocol" (see the ITSOL directory of the package) and provides a simple and effective interface for integrating the iterative subroutines with a user-customized matrix-vector product procedure. Specifically, the Krylov solver places at the interface a vector to take the user-provided matrix-vector product during the iteration process. This simple data-

passing interface circumvents the potential trouble or limitation in parameter-passing interface.

The convergence rate of a Krylov subspace method depends in part on the condition number of the matrix A. In AFMPB, A is well conditioned as a result of the BIE reformulation (2.4). In fact, A is the sum of the identity operator and a compact operator with well-bounded singular values. Preliminary numerical experiments show that the full GMRES method converges numerically in fewer iterations than the other methods, in agreement with analytical expectations. However, the memory requirement by the GMRES method increases linearly with k, the number of iterations. The reorthogonalization at step k requires k matrix-vector multiplications, making the total number of matrix-vector multiplications grow quadratically with k. In other words, GMRES procedure becomes very costly in both memory space and arithmetic operations. To control such cost growth, one can restart GMRES every k_0 steps, with a prescribed parameter $k_0 \ll N$. The restarted version is denoted as GMRES(k_0). The convergence rate with BiCGStab or TFQMR is slightly slower than that of the full GMRES. But the memory requirement for BiCGStab or TFQMR is independent of k, and the total number of matrix-vector multiplications grows linearly with k. In general, the choice of a Krylov iterative solver is application-specific. The default choice of the Krylov subspace solver in AFMPB is a restarted version of GMRES [8].

The GMRES method has been used in previous BIE based LPB solvers [17]. Further improvements in computational efficiency may be achieved by numerically preconditioning the linear system resulted from the Fredholm second-kind integral equation formulation, and by employing parallel versions of the Krylov subspace methods.

2.6 Adaptive fast multipole methods

AFMPB uses the new version of the adaptive fast multipole methods (FMMs) [29,31] for fast matrix-vector multiplications in the iterative methods. The arithmetic complexity of each matrix-vector product using FMM is $\mathcal{O}(N)$. The number of iterations is nearly independent of N with the Krylov methods. Therefore the discretized linear system can be solved with $\mathcal{O}(N)$ arithmetic operations.

An ingenious idea behind the FMM is to represent the matrix in a compressed form and further utilize it in the matrix-vector product. In this compressed form, submatrices are associated with interactions of regions at differential spatial scales. Submatrices corresponding to relatively far-field interactions are expressed in numerical low-rank form, while those for near-field interactions are in sparse forms; see the matrix interpretation of the FMM from Sun and Pitsianis [57].

AFMPB uses the compressed form derived from the *multipole expansions* for the Coulomb potential (Poisson equation) of particles located at ρ_i carrying a charge q_i, $i=1,\cdots,N$,

$$\phi(R,\theta,\psi)=\sum_{i=1}^{N} q_i \cdot \frac{1}{|\vec{R}-\vec{\rho}_i|} \approx \sum_{n=0}^{P} \sum_{m=-n}^{m=n} M_n^m \frac{Y_n^m(\theta,\psi)}{R^{n+1}}, \tag{2.5}$$

and the multipole coefficients are computed as follows

$$M_n^m = 8\sum_{i=1}^{N} q_i \cdot \rho_i^n \cdot Y_n^{-m}(\alpha_i,\beta_i). \tag{2.6}$$

In the formulas, P is the length of the expansion, determined by an accuracy requirement, (R,θ,ψ) and $(\rho_i,\alpha_i,\beta_i)$ are the spherical coordinates of the "far-field" target (evaluation) point $\vec{R}$ and source point $\vec{\rho}_i$, respectively, the origin is assumed to be the center of the source points, Y_n^m is the spherical harmonics function of order n and degree m, specified by

$$Y_n^m(\theta,\psi) = \sqrt{\frac{(2n+1)(n-|m|)!}{4\pi(n+|m|)!}} \cdot P_n^{|m|}(cos\theta)e^{im\psi}, \tag{2.7}$$

with P_n^m as the associated Legendre polynomial [10].

A similar expansion for the Debye-Hückel (screened Coulomb) interactions is derived as follows,

$$\phi(R,\theta,\psi) = \sum_{i=1}^{N} q_i \cdot \frac{e^{-\kappa|\vec{R}-\vec{\rho}_i|}}{|\vec{R}-\vec{\rho}_i|} \approx \sum_{n=0}^{P}\sum_{m=-n}^{m=n} M_n^m \cdot k_n(\kappa R)\cdot Y_n^m(\theta,\psi), \tag{2.8}$$

where the multipole coefficients are computed by

$$M_n^m = 8\kappa\sum_{i=1}^{N} q_i \cdot i_n(\kappa\rho_i)\cdot Y_n^{-m}(\alpha_i,\beta_i). \tag{2.9}$$

Here, $i_n(r)$ and $k_n(r)$ are the modified spherical Bessel and modified spherical Hankel functions, respectively. They are related to the conventional Bessel function as follows [10],

$$i_n(r) = \sqrt{\frac{\pi}{2r}} I_{n+1/2}(r), \quad k_n(r) = \sqrt{\frac{\pi}{2r}} K_{n+1/2}(r),$$

with

$$I_v(r) = \imath^{-v} J_v(\imath r), \quad K_v(r) = \frac{\pi}{\sin v\pi}[I_{-v}(r) - I_v(r)].$$

In the algorithmic aspect, FMM shares some common features with the so-called tree-code algorithms. Both partition the space hierarchically, using an oct-tree data structure, and both have intrascale translations. The arithmetic complexity of tree-code algorithms, such as that by Appel [11] or by Barnes and Hut [12], is $\mathcal{O}(N\log N)$. The prefactor depends on the detailed specification of the interaction list regions at each and every level. In any case, there are $\mathcal{O}(\log N)$ scale levels. By the Cartesian box setting as in the FMM, at every level, each *particle* interacts with 189 boxes in its "interaction list" via intrascale translation in a tree-code algorithm. Each translation follows the multipole expansion of $\mathcal{O}(P^2)$ terms. The FMM by Greengard and Rokhlin [30], introduced for the first time the local expansions, intrascale multipole-to-local translations and interscale multipole-to-multipole, local-to-local translations, each equipped with rigorous approximation analysis. In local expansions, the coordinate systems are with respect to the target points.

The multipole-to-local translations at each scale level are among boxes, instead of from boxes to particles. The source particle information is first aggregated into boxes at the bottom (particle) level and will be further aggregated to the higher levels, up to the top, during multipole-to-multipole translations. In the local-to-local translations, the interaction information is accumulated from the top, level by level, and disaggregated into child boxes. In short, each particle interacts with the other particles in the near-field boxes only at the bottom level. The remaining interactions and translations are carried out through the boxes. Notice that the total number of boxes is $O(N)$. As a result, the arithmetic complexity of the FMM becomes linear with N. Specifically, the local expansion for Coulomb interactions is as follows,

$$\phi(R,\theta,\psi)=\sum_{i=1}^{N} q_i\cdot\frac{1}{|\vec{R}-\vec{\rho}_i|}\approx\sum_{n=0}^{P}\sum_{m=-n}^{m=n} L_n^m\cdot R^n Y_n^m(\theta,\psi), \tag{2.10}$$

where L_n^m are the local expansion coefficients. A similar expansion for the screened Coulomb interaction can be described as

$$\phi(R,\theta,\psi)=\sum_{i=1}^{N} q_i\cdot\frac{e^{-\kappa|\vec{R}-\vec{\rho}_i|}}{|\vec{R}-\vec{\rho}_i|}\approx\sum_{n=0}^{P}\sum_{m=-n}^{m=n} L_n^m\cdot i_n(\kappa R)\cdot Y_n^m(\theta,\psi). \tag{2.11}$$

However, the prefactor in the complexity of the original FMM is larger than $\log N$ for systems of size N in practical range. The prefactor is substantially reduced in what is referred to as the "new version of the FMM" [31]. It is achieved by (1) introducing exponential expansions to diagonalize the multipole-to-local translation, and (2) using a "merge-and-shift" technique to reduce the total number of intrascale translations. The new version becomes highly competitive in systems of practical range. Numerical experiments show that the new version of FMM for the screened Coulomb interaction breaks even with the direct calculation when the number of particles N is as small as 500 when only 3-digit accuracy is required, or when N is 1000 for 6-digit accuracy requirement. In the AFMPB solver, the new version of FMMs are implemented for the Laplace and LPB equations to calculate the electrostatic interactions efficiently, with certain routines adopted from FMMSuite [9]. It also adapts to various geometries and sample distributions. Detailed descriptions of the adaptive new version FMM can be found elsewhere [20, 29, 31, 34].

We will comment in Section 3 on the comparison between FMM-based algorithms and FFT-based algorithms in biosystem simulations.

2.7 Parallel traversal of spatio-temporal graphs

We have introduced the key algorithmic components in the previous sections, and we have released our first sequential version of the AFMPB solver under open source license agreement [42]. We now turn our discussion to parallel computation. Since the arithmetic complexities have been substantially reduced toward the theoretically minimal,

the hope and effort to further speed up the computation rest mostly on parallelization. Fortunately, we have available advanced computer architectures, especially multicore processors, graphics processing units and their interconnections. In this section, we first introduce how the computation associated with the FMM, as a nontrivial example, can be analyzed and orchestrated as traversing a spatio-temporal directed acyclic graph (ST-DAG) so that its parallelization can be studied systematically. We then introduce in the next section a dynamic prioritization scheme toward parallel traversing the graph in minimal time.

The FMM graph may be better described by its subgraphs and associated operations. The FMM tree is familiar to many. In fact, it is the fusion of a source tree and a target tree, representing two spatial partitions in one spatial domain. One partitions the source ensemble, and the other partitions the target ensemble; see Fig. 3 for a view of the source and target trees in separation, shown in red and blue colors respectively. The *spatial nodes* (vertices) of the graph include the sets of source and target boxes in the FMM tree at levels $\ell=0,\cdots,\ell_{\max}$, denoted by $\mathcal{V}_\ell^s$ and $\mathcal{V}_\ell^t$, respectively. A spatial node $\mathcal{V}_i$ may be visited multiple times during a computation process, such as in an iterative or time-marching process. We further unfold such a spatial node into multiple *spatio-temporal* nodes $\mathcal{V}_{ik}=(\mathcal{V}_i,k)$, specifying the kth visit to the spatial node $\mathcal{V}_i$. The spatio-temporal nodes are connected by *directed edges* indicating the dependency from a predecessor node to a successor node. The nodes without predecessors or successors are respectively referred to as the *frontier* or *terminal* nodes, the rest are called *interior* nodes. We denote by $\mathcal{G}$ the ST-DAG consisting of the spatio-temporal nodes and the directed edges.

In the FMM, the computation traverses the graph $\mathcal{G}$ as follows. (1) Upward traversal on the source tree, up to level 2. Every leaf node in the source tree computes and renders a vector-valued output data, namely, the multipole expansion coefficients. A nonleaf node accumulates and translates data from its child nodes into its own multipole coefficients. Such upward interscale operations are known as the multipole-to-multipole translations, denoted by T_{MM}; see Fig. 3. (2) The multipole-to-local translations T_{ML} for the "interaction list" boxes, which connects the source and target nodes in the ST-DAG $\mathcal{G}$. We denote the corresponding bipartite subgraphs by $\mathcal{G}_\ell$ at each level $\ell\in[2,\ell_{\max}]$. The vertex set of $\mathcal{G}_\ell$ includes $\mathcal{V}_\ell^s$ and $\mathcal{V}_\ell^t$, the respective (disjoint) sets of the source and target boxes in the FMM tree at level ℓ; the edge set of $\mathcal{G}_\ell$ represents the source-target interaction. A source box $\mathcal{B}_s$ and a target box $\mathcal{B}_t$ at level ℓ are connected by an edge if (a) $\mathcal{B}_s\cap\mathcal{B}_t=\emptyset$, and (b) their parents are near-neighbors, i.e., with common boundary; see Fig. 3. Notice that the interaction subgraphs $\mathcal{G}_\ell$ at different levels are also disjoint from each other. If the source and target ensembles are uniformly distributed, $\mathcal{G}_{\ell+1}$ has 2^d times as many nodes as $\mathcal{G}_\ell$ does, where d is the dimensionality of the problem. (3) Downward traversal of the target tree. At level $\ell>2$, every node gets the local coefficients from its parent, merges them with its own at level ℓ, and makes the integrated results available to its child nodes or residing target locations. These downward interscale operations are known as the local-to-local translations, denoted by T_{LL}; see Fig. 3.

In the remainder of this section, we shall briefly review previous efforts in paralleliz-

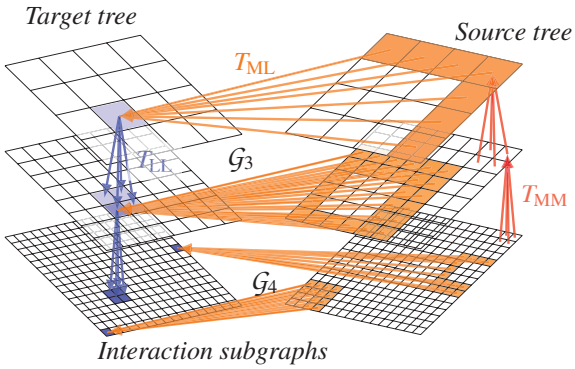

Figure 3: Example of ST-DAG for the FMM.

ing the FMM. The first study is by Greengard and Gropp with the Laplacian kernel on uniformly distributed data [28]. It showed that the overall time complexity was $O(\log N)$ using $O(N)$ processors and predicted that shared memory systems were more suitable. Many other methods attempted to partition and balance the computation work among the available processing elements (PEs). They may be categorized into the *tree node decomposition* and *spatial decomposition* methods. Tree decomposition approach is based on the FMM tree: one estimates the computation work at each tree node, and distributes it evenly across the available PEs. This approach is implemented by using either the cost-zone technique on shared memory architectures [55,56] or the hashed oct-tree technique on distributed memory architectures [60]. By assuming an upper bound on the partition level, Teng gave a provably good partition algorithm with linear complexity [58]. The distribution independent adaptive tree (DIAT) [54] is developed to handle the extreme case where the partition level becomes $\mathcal{O}(N)$. It integrates a compressed oct-tree and a balanced binary search tree. The spatial decomposition approach divides the problem domain into non-overlapping subregions, while the tree nodes at two different PEs by the tree node decomposition approach may spatially overlap. Each spatial subdomain contains an approximately equal number of particles and is assigned to an available PE. For instance, Salmon used the orthogonal recursive bisection (ORB) technique [53]. At each PE, a locally essential tree (LET) is constructed, with the data required for the computation on the particles owned by the PE.

2.8 Dynamic prioritization

We introduce in this section a dynamic prioritization scheme for traversing a general ST-DAG in parallel, which is a significant departure from previous ad-hoc approaches. Due to the page space limitation, we give only a sketch of the main idea. The paper by Zhang et al. [63] introduced the concept and scheme for parallel scheduling in general. We show in Section 3 the particular application of this scheme to parallel FMM and its remarkable performance.

We start with the constraint-free situation where there are infinite or sufficient computing resources. The shortest parallel traversal time in this case can be achieved by following a simple data-driven rule. The frontier nodes start their computation immediately and unconditionally, which are removed together with their incident edges upon the completion of computations. In the reduced ST-DAG, nodes with available input data become the new frontier nodes and start their computations, and so on. The latest time step among all the terminal nodes in the original ST-DAG is the absolute shortest time. The predecessors of the terminal nodes in the final step are time critical. Within the same period, however, it is not necessary to start every noncritical node at the earliest possible time. We define the absolute ranking as follows. Adopting Hu's methodology [33], we introduce an artificial sink node to the graph under consideration as the sole successor to all the original terminal nodes. The absolute rank of a node v is the length of the longest path connecting v and the sink node. With this ranking system, one can invoke the nodes of the highest rank among the frontier nodes at any given time while maintaining the shortest time in parallel traversal. We also define the unconditionally sufficient number of PEs as the maximal number of nodes in concurrent computation at any time, across all possible schedules for parallel traversing in the absolute shortest time.

The minimization in traversal time becomes complicated when resources are limited, such as by the number of PEs and variations in latency due to memory accesses, thread switches, time sharing, etc. The problem may be further complicated if the graphs are highly irregular. To this end, we introduce the concept of *adaptive ranking* among the frontier nodes. Assume the resource constraint first, where the number of available PEs, N_{PE}, is far smaller than sufficient for the absolute shortest time case. At any step, when there are no more than N_{PE} frontier nodes, the frontier nodes are served all at once, just as the constraint-free situation; otherwise, the nodes of higher ranks are served first. Consider the case when two frontier nodes u and v are of the same absolute rank, but only one of them can be served with the rest $N_{PE}-1$ nodes of higher ranks. A random tiebreaker is a candidate policy for utilizing the parallel capacity at the current step. However, it is not an optimal policy for any DAG toward the shortest time under resource constraints. It is necessary to look ahead and maximize the number of frontier nodes at the next steps, which influence the subsequent steps. We want to maintain the maximal degree of concurrency at any and all steps. We are also concerned with the execution dynamics. Therefore, we associate each frontier node c with an order pair, $\langle d_m^-(c), n_s(c)\rangle$,

$$d_m^-(c) = \min_{(c,d)\in\mathcal{E}(\mathcal{G})} \deg^-(d), \tag{2.12}$$

where (c,d) is a directed edge from c to d in the edge set $\mathcal{E}(\mathcal{G})$, and $\deg^-(d)$ is the in-degree at node d. When $d_m^-(c)=1$, $n_s(c)$ equals to 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 to the out-degree of c, $\deg^+(c)$. The completion of c makes its successors one step closer to the front. Back to the selection between equal-

ranked nodes u and v for service by one available PE. We give u a higher priority if

$$\langle d_m^-(u), -n_s(u)\rangle < \langle d_m^-(v), -n_s(v)\rangle \tag{2.13}$$

in the lexicographical order. A random selection breaks the tie in (2.13). Suppose u is chosen. Upon u's completion, each of its direct successors decrements its in-degree by one. The ranking criterion (2.13) may seem counterintuitive in the case $d_m^-(u) = 1$, as opposed to the other case $d_m^-(u) > 1$. They are consistent in the sense that we deal in the former case with immediate node release in order to maximize the concurrency at the very next step, and in the latter case, with the greatest reduction in parallel dependency for maximizing the concurrency in future steps.

We also remark that the problem of traversing a DAG in parallel in the shortest time, with the number of PEs constant or varying, is in general NP-complete [59].

3 Experimental results

3.1 Electrostatic calculations

We describe numerical experiments for electrostatic calculations of a protein system with the AFMPB solver as well as with the APBS solver [4]. APBS is popularly used. It assumes the solvent excluded surface (SES) [50] by default, uses internal implicit mesh generation and employs a multigrid finite element method. AFMPB admits meshes generated by the new tool TMSmesh or other popularly used mesh generators, and employs Krylov subspace methods for solving the discretized BIEs with the interface conditions described earlier. As the two solvers differ in all the major factors, the differences in computing requirements and numerical results provide a comprehensive perspective to their comparisons, in addition to the arguments or analysis on each of the computational components.

In order to single out the impact by an individual factor on numerical accuracy and computational efficiency in numerical solutions, we describe four sets of experiments, some of which we completed very recently and the others were carried out earlier by Lu, one of the co-authors, and his other collaborators. The first set of experiments demonstrate the impact by the difference between the surface descriptions and mesh generators on the accuracy in calculated solvation energy. In particular, AFMPB is used with two types of meshes: meshes generated by MSMS with a solvent excluded surface and meshes generated by TMSmesh with a Gaussian surface. In order to observe the response in numerical behaviors as the mesh resolution becomes finer or coarser, the experiments shall be carried out at different mesh resolutions. To this end, we restricted the experiments to the study of small molecules. Lu and his colleagues had reported the results of such experiments in a very recent paper [19] on a few small molecules, including ADP. They found that with each of the two types, the total surface area of the generated mesh converges as the mesh resolution becomes sufficiently fine, so do the enclosed volume

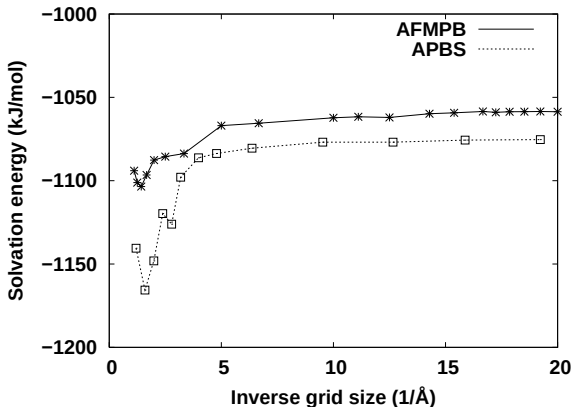

Figure 4: The convergence behaviors of AFMPB and APBS in energy calculations. The grid size of AFMPB denotes the side length of the smallest grid box used by TMSmesh to generate the triangulated molecular Gaussian surface [19].

and the solvation energy. They also observed a small discrepancy (1%) in the converged values in the calculated energy between the two mesh types.

We continued their efforts with ADP. In the second set of experiments, we compared the calculated solvation energies by AFMPB with TMSmesh to that by APBS, which allows the user to specify the mesh resolution. As the mesh resolution becomes finer, the calculated energy values by both solvers show similar convergence behaviors, as shown in Fig. 4. APBS renders sufficiently accurate results when the grid resolution is about 0.2 $\sim$ 0.25 Å. Fig. 4 shows that AFMPB reaches the same accuracy at about the same grid resolution. Fig. 4 also shows the discrepancy between the converged values, which remains small (1%). The experiments of the third set are to provide a comparison in computational efficiency. Instead of making direct comparisons, we refer the reader to the paper by Lu et al. [43], where they reported that the nonadaptive counterpart of AFMPB outperformed APBS already on a few systems that are not large. We add only that AFMPB is more efficient than its nonadaptive counterpart.

In the fourth set of experiments, we were concerned in particular with the impact of the molecule size and complexity on the effectiveness of the AFMPB solver. We used a larger molecule, the proton-gated ion channel GLIC, taken from a MD simulation trajectory [25]. This high-resolution crystal structure of GLIC captured in a potentially open state may provide detailed atomic-level insights into ion conduction and selectivity mechanisms in the ion channels. GLIC consists of 25,300 atoms, in 100 Å$\times$ 80 Å$\times$ 128 Å dimensions. In our experiments, the molecular surface mesh generated by MSMS, with $1/\text{Å}^2$ density and 1.5 or 1.4 Å probe radius, contained many isolated nodes. Such topological faults inhibited successful execution of AFMPB. In comparison, the molecular surface generated by the TMSmesh, with 329,764 triangular elements and 164,523 nodes (see Fig. 1), allowed a successful AFMPB execution, returned with a solvation energy of -11053.5 kcal/mol. The computed surface potential is shown in Fig. 5.

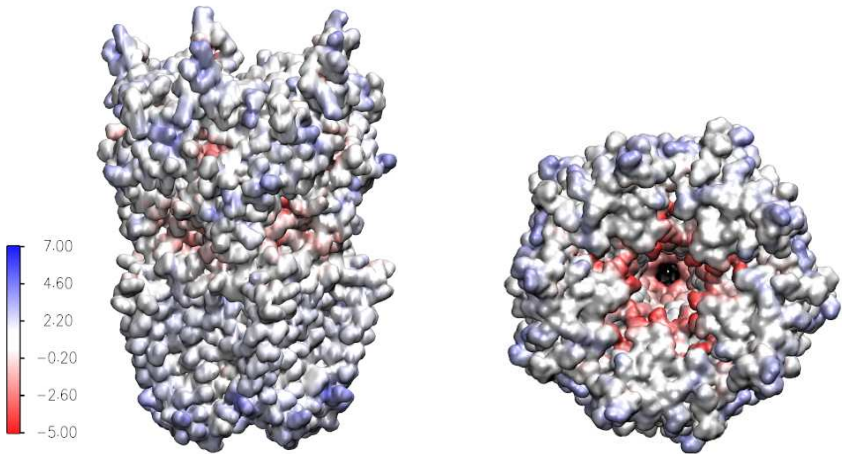

Figure 5: Side and top views of the surface electrostatic potential (in unit kcal/mol·e) of an ion channel protein, GLIC (PDB code: 3ehz).

3.2 Parallel FMM in AFMPB

We have recently integrated the parallel FMM into a parallel version of the AFMPB under development. We present here preliminary experimental results to demonstrate the efficiency of the parallel FMM. The experiments were carried out on a 24-core workstation with 64 GB memory. We used the system function *pthread_set_affinity_np* to select and activate a partial or the entire set of cores for scalability evaluations. For the numerical task, we computed the pairwise interactions of N particles governed by the screened Coulomb potential $e^{-\kappa \mathbf{r}}/\mathbf{r}$, where $\mathbf{r}$ is the Euclidean distance between two interacting particles and κ is a modest positive constant. Particles were randomly distributed within a normalized cube in $\mathbb{R}^3$.

We show in Fig. 6 the parallel performance in terms of the weak scaling in the top

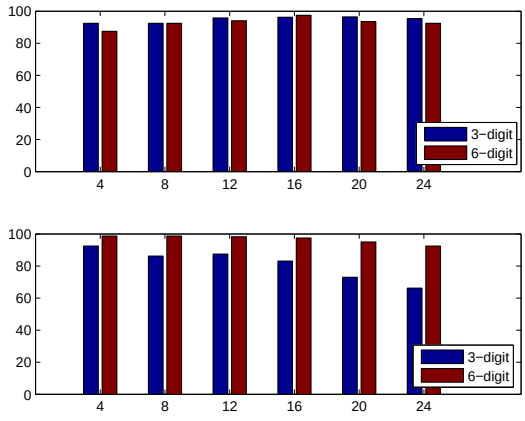

Figure 6: Parallel FMM's parallel efficiency in weak scaling (top chart) and strong scaling (bottom chart).

chart and the strong scaling in the bottom chart. The ordinate axes in both bar charts stand for the parallel efficiency, defined as $\frac{T_s}{N_{PE}\cdot T_p(N_{PE})}$, where T_s is the runtime of the best sequential algorithm, N_{PE} is the number of cores, and T_p is the runtime of the best parallel algorithm on N_{PE} cores. The blue and red bars display the results of 3-digit and 6-digit accuracy, respectively. In weak scaling tests, we kept the ratio between N and N_{PE} constant. Specifically, the ratio was $3.75e6$ for 3-digit accuracy and $6.25e5$ for 6-digit accuracy. The parallel efficiency remains above 90% in each case, in the presence of thread management overheads. Without dynamic priority scheduling, the parallel efficiency was above 90% up to 8 cores, but dropped to 60% with 24 cores. In strong scaling tests, the problem size N was kept constant at $1.5e7$. The parallel efficiency remains above 90% for 6-digit case since it involves more numerical operations. The 3-digit case lacks computational work to keep all 24 cores busy most of the time. This is in part caused by the aggregation operations to counterbalance the overhead in thread management. The balance between the granularity in concurrent operations and the overhead in thread management is delicate.

3.3 Additional discussion

Finally, we shall address, if not fully, the issue on the comparison between the FMM-based algorithms and FFT-based algorithms in biosystem simulations. Methods from the latter category include the precorrected FFT (pFFT) method [49] and particle mesh Ewald (PME) technique [22]. To shed a light on the fundamental, we made an abstract experiment setting, keeping only the most dominant algorithmic component. We consider two types of data: cubic volume data and spherical surface data. There are two typical components in the FFT-based algorithms: the interpolations and the FFTs. By the interpolations the data are translated to and from a Cartesian grid, before and after the FFTs, respectively. We keep the FFTs only. However, when the data are on the surface, we take into consideration of the change in data types made by the interpolations or required by the FFT. That is, the data on the surface are translated into the volume data in a cube. For the FMM-based algorithms, we keep the FMM only, which admits data at arbitrary locations. We assume, as in most of other FMM applications, that the FMM translators are predetermined numerically.

The experiments were carried out on the aforementioned workstation. The FFT experiments used the best schedule in the FFTW; the FMM code, written in C, was compiled using gcc without linking any existing high-performance library, without loop-level unrolling or tuning. The timing results (in seconds) are summarized in Table 1, where N is the data size, T_{FFT} is the time spent on two FFTs (forward and backward), and T_{FMM} is the time spent on the FMM. With cubic volume data, the FFT is the front winner. With spherical surface data, the FMM surpasses the FFT at three-quarter million size. This is due to the change of surface data to volume data at the FFT step, i.e., the FFT operates on the data of size 512^3 and 1024^3, respectively, for the two surface data sets. We also note that the FMM takes less time on the surface data of 3 million size than the volume data

Table 1: Basic comparisons in execution time (seconds) between the FFT-based and FMM-based methods for calculating electrostatic interactions with cubic volume and spherical surface data.

Data distributions	$N \times 2^{20}$	T_{FFT}	T_{FMM}
Cubic volume	2.00	0.3	13.1
	16.00	0.4	14.9
Spherical surface	0.75	6.4	2.3
	3.00	30.0	9.0

of 2 million. This manifests the adaptation of the FMM to data distribution. These tell us that the FFT-based algorithms are likely more efficient when particles are uniformly distributed in a 3D cubical domain. The algorithms that are based on the adaptive FMM are perhaps more efficient when particles are distributed on a 2D surface as in the case when we use the BIE method or other situations where the data are highly clustered.

These comparisons, however, may change dramatically on distributed memory architectures. The distributed FFT is known to suffer from high latency in synchronized interprocess communications. The FMM potentially allows asynchronized communications.

Acknowledgments

The AFMPB solver uses many open-source packages, including SPARSKIT [8], MSMS [5], FMMSuite [9], and several numerically important subroutines for the new-version of the FMM provided by Greengard and Rokhlin. We thank our colleagues and collaborators in computer science, mathematics and bioscience for their contributions and suggestions at every stage of the AFMPB development. This work was supported by NSF, DOE, HHMI, and NIH (B. Z./X. S./N. P.: NSF 0905164, B. Z./J. H.: NSF 0811130 and NSF 0905473, J. A. M.: NSF MCB1020765 and NIH GM31749) and the NSF Center of Theoretical Biological Physics (CTBP). Additionally, B. L. is partially funded by the Chinese Academy of Sciences, the State Key Laboratory of Scientific/Engineering Computing, and the China NSF (NSFC1097218). X. C. is supported in part by the U.S. Department of Energy Field Work Proposal ERKJE84. We also thank the reviewers for their valuable comments and suggestions.

References

[1] `http://wiki.c2b2.columbia.edu/honiglab_public/index.php/Software:DelPhi`.
[2] `http://wiki.c2b2.columbia.edu/honiglab_public/index.php/Software:GRASP`.
[3] `http://mccammon.ucsd.edu/uhbd.html`.
[4] `http://www.poissonboltzmann.org/apbs`.
[5] `http://mgltools.scripps.edu/packages/MSMS/`.
[6] `http://fetk.org/codes/gamer/`.

[7] http://lsec.cc.ac.cn/~lubz/Meshing.html.
[8] http://www-users.cs.umn.edu/~saad/software/SPARSKIT/index.html.
[9] http://www.fastmultipole.org/.
[10] M. Abramowitz and I. A. Stegun. Handbook of Mathematical Functions. Dover, 1970.
[11] A. W. Appel. An efficient program for many-body simulation. SIAM Journal on Scientific and Statistical Computing, 6:85–103, 1985.
[12] J. Barnes and P. Hut. A hierarchical $O(N\log N)$ force-calculation algorithm. Nature, 324:446–449, 1986.
[13] D. Bashford. An object-oriented programming suite for electrostatic effects in biological molecules. Lecture Notes in Computer Science, 1343:233–240, 1997.
[14] P. W. Bates, G. W. Wei, and S. Zhao. Minimal molecular surfaces and their applications. Journal of Computational Chemistry, 29:380–391, 2008.
[15] A. Bondi. van der Waals volumes and radii. The Journal of Physical Chemistry, 68:441–451, 1964.
[16] A. H. Boschitsch and M. O. Fenley. Hybrid boundary element and finite difference method for solving the nonlinear Poisson-Boltzmann equation. Journal of Computational Chemistry, 25:935–955, 2004.
[17] A. H. Boschitsch, M. O. Fenley, and H. X. Zhou. Fast boundary element method for the linear Poisson-Boltzmann equation. The Journal of Physical Chemistry B, 106:2741–2754, 2002.
[18] D. A. Case, T. E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The Amber biomolecular simulation programs. Journal of Computational Chemistry, 26:1668–1688, 2005.
[19] M. X. Chen and B. Z. Lu. TMSmesh: A robust method for molecular surface mesh generation using a trace technique. Journal of Chemical Theory and Computation, 7(1):203–212, 2011.
[20] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. Journal of Computational Physics, 155:468–498, 1999.
[21] D. D'Agostino, A. Clematis, I. Merelli, L. Milanesi, and M. Coloberti. A grid service based parallel molecular surface reconstruction system. In Proceedings of 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pages 455–462, 2008.
[22] T. Darden, D. York, and L. Pedersen. Particle mesh Ewald: an $N\log N$ method for Ewald sums in large systems. Journal of Chemical Physics, 98:10089–10092, 1993.
[23] J. Dzubiella, J. M. J. Swanson, and J. A. McCammon. Coupling hydrophobicity, dispersion, and electrostatics in continuum solvent models. Physics Review Letters, 96:087802, 2006.
[24] F. Fogolari, A. Brigo, and H. Molinari. The Poisson-Boltzmann equation for biomolecular electrostatics: a tool for structural biology. Journal of Molecular Recognition, 15:377–392, 2002.
[25] S. Fritsch, I. Ivanov, H. L. Wang, and X. L. Cheng. Ion selectivity mechanism in a bacterial pentameric ligand-gated ion channel. Biophysical Journal, 100:390–398, 2011.
[26] A. Gouaillard, A. Gelas, and S. Megason. Triangular meshes Delaunay conforming filter. Insight Journal, July-December, 2008.
[27] J. A. Grant, B. T. Pickup, and A. Nicholls. A smooth permittivity function for Poisson-Boltzmann solvation methods. Journal of Computational Chemistry, 22:608–640, 2001.
[28] L. Greengard and W. Gropp. A parallel version of the fast multipole method. Computers & Mathematics with Applications, 20:63–71, 1990.
[29] L. Greengard and J. Huang. A new version of the fast multipole method for screened Coulomb interactions in three dimensions. Journal of Computational Physics, 180:642–658, 2002.

[30] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. Journal of Computational Physics, 73:325–348, 1987.
[31] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numerica, 6:229–269, 1997.
[32] M. J. Hsieh and R. Luo. Physical scoring function based on AMBER force field and Poisson-Boltzmann implicit solvent for protein structure prediction. Proteins, 56:475–486, 2004.
[33] T. C. Hu. Parallel sequencing and assembly line problems. Operations Research, 9:841–848, 1961.
[34] J. Huang, J. Jia, and B. Zhang. FMM-Yukawa: an adaptive fast multipole method for screened Coulomb interactions. Computer Physics Communications, 180:2331–2338, 2009.
[35] W. Im, D. Beglov, and B. Roux. Continuum solvation model: computation of electrostatic forces from numerical solutions to the Poisson-Boltzmann equation. Computer Physics Communications, 111:59–75, 1998.
[36] S. Jo, M. Vargyas, J. Vasko-Szedlar, B. Roux, and V. Im. PBEQ-solver for online visualization of electrostatic potential of biomolecules. Nucleic Acids Research, 36:270–275, 2008.
[37] A. H. Juffer, E. F. F. Botta, B. A. M. Vankeulen, A. Vanderploeg, and H. J. C. Berendsen. The electric potential of a macromolecule in a solvent: a fundamental approach. Journal of Computational Physics, 97:144–171, 1991.
[38] S. Kuo, M. Altman, J. Bardhan, B. Tidor, and J. White. Fast methods for biomolecule charge optimization. In Proceedings of International Conference on Modeling and Simulation of Microsystems, 2002.
[39] S. Kuo and J. White. A spectrally accurate integral equation solver for molecular surface electrostatics. In Proceedings of the IEEE Conference on Computer-Aided Design, 2006.
[40] B. Lee and F. Richards. The interpretation of protein structures: estimation of static accessibility. Journal of Molecular Biology, 55:379–400, 1971.
[41] J. Liang and S. Subramaniam. Computation of molecular electrostatics with boundary element methods. Biophysical Journal, 73:1830–1841, 1997.
[42] B. Lu, X. Cheng, J. Huang, and J. A. McCammon. AFMPB: an adaptive fast multipole Poisson-Boltzmann solver for calculating electrostatics in biomolecular systems. Computer Physics Communications, 181:1150–1160, 2010.
[43] B. Lu, X. Cheng, and J. A. McCammon. "New-version-fast-multipolemethod" accelerated electrostatic calculations in biomolecular systems. Journal of Computational Physics, 226:1348–1366, 2007.
[44] B. Lu and J. A. McCammon. Improved boundary element methods for Poisson-Boltzmann electrostatic potential and force calculations. Journal of Chemical Theory and Computation, 3:1134–1142, 2007.
[45] B. Lu, Y. C. Zhou, G. A. Huber, S. D. Bond, M. J. Holst, and J. A. McCammon. Electrodiffusion: a continuum modeling framework for biomolecular systems with realistic spatiotemporal resolution. Journal of Chemical Physics, 127:135102, 2007.
[46] J. D. Madura, J. M. Briggs, R. C. Wade, M. E. Davis, B. A. Luty, A. Ilin, J. Antosiewicz, M. K. Gilson, B. Bagheri, L. R. Scott, and J. A. McCammon. Electrostatics and diffusion of molecules in solution - simulations with the University of Houston Brownian Dynamics Program. Computer Physics Communication, 91:57–95, 1995.
[47] A. Nicholls, K. A. Sharp, and B. Honig. Protein folding and association: insights from the interfacial and thermodynamic properties of hydrocarbons. Proteins: Structure, Function, and Bioinformatics, 11:281–296, 1991.
[48] M. Orozco and F. Lugue. Theoretical methods for the description of the solvent effect in

biomolecular systems. Chemical Reviews, 100:4187–4226, 2000.
[49] J. R. Phillips and J. K. White. A precorrected-FFT method for electrostatic analysis of complicated 3D structures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16:1059–1072, 1997.
[50] F. M. Richards. Areas, volumes, packing and protein structure. Annual Review in Biophysics and Bioengineering, 6:151–176, 1977.
[51] V. Rokhlin. Solution of acoustic scattering problems by means of second kind integral equations. Wave Motion, 5:257–272, 1983.
[52] B. Roux and T. Simonson. Implicit solvent models. Biophysical Chemistry, 78:1–20, 1999.
[53] J. Salmon. Parallel Hierarchical N-Body Methods. PhD thesis, California Institute of Technology, 1990.
[54] F. E. Sevilgen and S. Aluru. A unifying data structure for hierarchical methods. In Proceedings of the ACM/IEEE Conference on Supercomputing, 1999.
[55] J. Singh, C. Holt, J. Hennessy, and A. Gupta. A parallel adaptive fast multipole method. In Proceedings of the ACM/IEEE Conference on Supercomputing, 1993.
[56] J. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity. Journal of Parallel and Distributed Computing, 27:118–141, 1995.
[57] X. Sun and N. P. Pitsianis. A matrix version of the fast multipole method. SIAM Review, 43:289–300, 2001.
[58] S. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. SIAM Journal on Scientific Computing, 19:635–656, 1998.
[59] J. D. Ullman. NP-complete scheduling problems. Journal of Computer and System Sciences, 10:384–393, 1975.
[60] M. Warren and J. Salmon. A parallel hashed oct-tree N-body algorithm. In Proceedings of the ACM/IEEE Conference on Supercomputing, 1993.
[61] J. Weiser, P. S. Shenkin, and W. C. Still. Optimization of Gaussian surface calculations and extension to solvent-accessible surface areas. Journal of Computational Chemistry, 20:688–703, 1999.
[62] Y. C. Zhou and M. Feig and G. W. Wei. Highly accurate biomolecular electrostatics in continuum dielectric environments. Journal of Computational Chemistry, 29:87–97, 2007.
[63] B. Zhang, J. Huang, N. P. Pitsianis, and X. Sun. Dynamic prioritization for parallel traversal of irregularly structured spatio-temporal graphs. In Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism, 2011.
[64] Y. Zhang, G. Xu, and C. Bajaj. Quality meshing of implicit solvation models of biomolecular structures. The Special Issue of Computer Aided Geometric Design on Applications of Geometric Modeling in the Life Sciences, 23:510–530, 2006.