Journal of Chemical Theory and Computation

An Adaptive Fast Multipole Boundary Element Method for Poisson−Boltzmann Electrostatics

Benzhuo Lu,*,† Xiaolin Cheng,‡ Jingfang Huang,§ and J. Andrew McCammon||

Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, People's Republic of China, Center for Molecular Biophysics, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Department of Mathematics, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina 27599-3250, Department of Chemistry & Biochemistry, Center for Theoretical Biological Physics, Department of Pharmacology, Howard Hughes Medical Institute, University of California, San Diego, California 92093

Received February 16, 2009

Abstract: The numerical solution of the Poisson−Boltzmann (PB) equation is a useful but a computationally demanding tool for studying electrostatic solvation effects in chemical and biomolecular systems. Recently, we have described a boundary integral equation-based PB solver accelerated by a new version of the fast multipole method (FMM). The overall algorithm shows an order *N* complexity in both the computational cost and memory usage. Here, we present an updated version of the solver by using an adaptive FMM for accelerating the convolution type matrix-vector multiplications. The adaptive algorithm, when compared to our previous nonadaptive one, not only significantly improves the performance of the overall memory usage but also remarkably speeds the calculation because of an improved load balancing between the local- and far-field calculations. We have also implemented a node-patch discretization scheme that leads to a reduction of unknowns by a factor of 2 relative to the constant element method without sacrificing accuracy. As a result of these improvements, the new solver makes the PB calculation truly feasible for large-scale biomolecular systems such as a 30S ribosome molecule even on a typical 2008 desktop computer.

1. Introduction

Electrostatic interactions play essential roles in many biological processes, such as enzymatic catalysis, molecular recognition, and bioregulation. Over the past three decades, the Poisson−Boltzmann (PB)-based continuum electrostatic calculation has become a common tool in theoretical studies of biomolecular systems such as proteins and DNAs in aqueous solutions. Many of these PB solvers rely on numerical solution of the PB equation. Among them, the PB solvers based on the finite difference methods, including DelPhi, GRASP, MEAD, UHBD, and the PBEQ,[1] have gained wide popularity, most likely due to their ease of implementation. A finite volume/multigrid PB solver APBS also enjoys increasing popularity over biochemistry and biophysical communities recently.[2,3] To our knowledge, APBS is the first program to enable distribution of PB calculations to a great number of processors, thus allowing extremely large-scale systems to be computed.

On the other hand, algorithms based on the boundary integral equation (BIE) approach have shown great promise for their efficiency on scaling and memory requirements.[4−6] These methods rely on Green's theorem and potential theory to recast the linear PB equation into a set of boundary integral equations that need to be solved only on the surface of the

* To whom correspondence should be addressed. E-mail: bzlu@lsec.cc.ac.cn.
† Chinese Academy of Sciences.
‡ Oak Ridge National Laboratory.
§ University of North Carolina at Chapel Hill.
|| University of California, San Diego.

molecule. Therefore, the number of unknowns is reduced relative to the volumetric discretization in finite difference and finite element methods. This surface integral equation idea is not new and was applied in the boundary element methods (BEM) in the early 1970s for different kinds of problems. Unfortunately most previous BEM implementations used Gauss eliminations to solve the resulting linear system. Even when a Krylov subspace based iterative solver was used for acceleration, the BEM approach was still limited by the cost associated with numerous surface integrations (matrix vector multiplications) that require an order $\sim N^2$ operations for a system with N surface elements. In the last 20 years or so, however, many fast algorithms have been introduced to efficiently evaluate these convolution type surface integrations, examples include the FFT-based algorithms (such as the precorrected FFT[7,8] and particle mesh Ewald methods[9,10]) and the multipole expansion-based techniques (such as the tree code[11,12] and fast multipole methods[13−17]). In particular, we want to mention our recent combination of the new version of the fast multipole method with the BEM formulation for PB equation, which has been shown numerically to be faster than existing PB solvers based on the finite-difference method for relatively large systems.[6]

However, our earlier implementation of the BEM/FMM approach for the PB equation adopts a nonadaptive tree structure for the sake of easy implementation, which is suitable for fairly uniform element distributions. For the surface integral equation formulation, as the elements distribute only on the surface of the molecule, at the lower levels of the tree structure a large number of boxes beyond the molecular surface are empty, which significantly compromises the computational efficiency of the algorithm because of the time and storage spent on these empty boxes. Moreover, the nonadaptive algorithm is more difficult to strike a load-balance between the number of elements in the local list (calculated directly) and those in the far-field (calculated using multipole and local expansions), thus further reducing its efficiency. By contrast, the adaptive FMM (AFMM) continues to subdivide boxes only until the number of elements in a box has reached a predefined number, thus creating a practically ‘uniform’ partition of particles in all childless boxes regardless of their sizes. In this paper, we present an improved implementation of the PB solver using an adaptive new version of FMM,[18] a “node-patch” discretization approach,[19] and the Krylov subspace iterative subroutines from the open source package SPARSKIT.[20] The resulting adaptive solver shows not only more efficient use of the memory but also significantly improves the load-balance between the local and far-field calculations, thus leading to faster calculation by several fold.

This paper is organized as follows. In Section 2, we describe the boundary integral equation formulation for the linearized PB solver. In Section 3, the “node-patch” discretization scheme is introduced to further reduce the number of unknowns. In Section 4, we discuss the Krylov subspace subroutines used in our solver, in particular, the package SPARSKIT and its convenient “reverse communication protocol”. In Section 5, we briefly discuss the adaptive new version of FMM. In Section 6, numerical results are presented to benchmark the efficiency and accuracy of the solver, and finally in Section 7, we conclude this paper and discuss how to further optimize the solver using optimized oct-tree structure based on “spectral graph theory”[21] and parallelization on multicore multiprocessor computers.

2. Boundary Integral Equation Formulations

The Poisson−Boltzmann equation takes its most standard form as

$$-\nabla(\varepsilon\nabla\phi) + \kappa^2 \sin h(\phi) = \sum_{i=1}^{M} q_i \delta(r - r_i) \tag{1}$$

When the electrostatic potential ϕ is small, the linearized PB equation can be obtained as

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

When Green’s second identity is applied, traditional boundary integral equations for the linearized PB equation for a single domain (molecule) can be written as,

$$\phi_p^{\text{int}} = \oint_s \left[G_{pt}\frac{\partial \phi_t^{\text{int}}}{\partial n} - \frac{\partial G_{pt}}{\partial n}\phi_t^{\text{int}}\right]\mathrm{d}S_t + \frac{1}{D_{\text{int}}}\sum_k q_k G_{pk} \qquad p, k \in \Omega \tag{3}$$

$$\phi_p^{\text{ext}} = \oint_s \left[-u_{pt}\frac{\partial \phi_t^{\text{ext}}}{\partial n} + \frac{\partial u_{pt}}{\partial n}\phi_t^{\text{ext}}\right]\mathrm{d}S_t \quad p \in \bar{\Omega} \tag{4}$$

where $\phi_p{}^{\text{int}}$ is the interior potential at position p of the molecular domain Ω, q_k is the kth source charge, and $S = \partial\Omega$ is the molecular boundary. There are a variety of ways to specify the molecular boundary (solute−solvent dividing surface), and it is known that different specifications of the boundary can lead to very different results (see, e.g., ref 22). This is a practically important issue but is beyond the scope of this work. The particular surface types used in this work will be noted in the later sections when encountered. $\phi_p{}^{\text{ext}}$ is the exterior potential at position p, D_{int} is the interior dielectric constant, t is an arbitrary point on the boundary, and n is the outward normal vector at t. In the formulas, G_{pt} and u_{pt} are the fundamental solutions of the corresponding Poisson and Poisson−Boltzmann equations, respectively. When point p approaches the surface S, by satisfying the boundary conditions $\phi^{\text{int}} = \phi^{\text{ext}}$ and $D_{\text{int}}(\nabla\phi^{\text{int}}n) = D_{\text{ext}}(\nabla\phi^{\text{ext}}n)$, eqs 3 and 4 become a set of self-consistent boundary integral equations (denoted as nBIEs),

$$\alpha_p f_p = \oint_s PV\left[\varepsilon G_{pt} h_t - \frac{\partial G_{pt}}{\partial n} f_t\right]\mathrm{d}S_t + \frac{1}{D_{\text{int}}}\sum_k q_k G_{pk} \qquad p \in S \tag{5}$$

$$(1 - \alpha_p) f_p = \oint_s \mathrm{PV}\left[-u_{pt} h_t + \frac{\partial u_{pt}}{\partial n} f_t\right]\mathrm{d}S_t\, p \in S \tag{6}$$

where PV denotes the principal value integral to avoid the singularity when $t \rightarrow p$, $f = \phi^{\text{ext}}$, $h = \nabla\phi^{\text{ext}}n$, and $\varepsilon = D_{\text{ext}}/D_{\text{int}}$. The coefficient constant α_p is 1/2 for a smooth surface, and more generally, it depends on the local surface geometry at node p. For a vertex of a polyhedron, the coefficient α_p equals $A_p/4\pi$, where A_p is the interior solid angle at p. The constant of 1/2 has been usually used in previous BEM/PB work, while we have recently demonstrated that the use of a geometry-dependent coefficient significantly improves the overall numerical accuracy for the potential evaluation.[19]

The derivative BIEs (dBIEs) can be obtained by linearly combining eqs 5 and 6 and their derivative forms (for smooth surface case).

$$\left(\frac{1}{2\varepsilon}+\frac{1}{2}\right)f_p = \oint_s \mathrm{PV}\left[(G_{pt}-u_{pt})h_t - \left(\frac{1}{\varepsilon}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)f_t\right]\mathrm{d}S_t + \frac{1}{D_{\mathrm{ext}}}\sum_k q_k G_{pk} \quad p \in S \quad (7)$$

$$\left(\frac{1}{2\varepsilon}+\frac{1}{2}\right)h_p = \oint_s \mathrm{PV}\left[\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\varepsilon}\frac{\partial u_{pt}}{\partial n_0}\right)h_t - \frac{1}{\varepsilon}\left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n}-\frac{\partial^2 u_{pt}}{\partial n_0 \partial n}\right)f_t\right]\mathrm{d}S_t + \frac{1}{D_{\mathrm{ext}}}\sum_k q_k \frac{\partial G_{pk}}{\partial n_0} \quad p \in S \quad (8)$$

where n is the unit normal vector at point t and n_0 is the unit normal vector at point p. The dBIEs lead to a well-conditioned (Fredholm second kind) system of algebraic equations. When Krylov subspace methods are applied to such systems, the number of iterations remains bounded even for a large number of elements. In our former work, we extended this form to systems of more than one separated molecules and provided a set of corresponding equations for force calculation.[6]

3. "Node-Patch" Discretization

After a typical triangular discretization, eqs 7 and 8 become

$$\left(\frac{1}{2\varepsilon}+\frac{1}{2}\right)f_p = \sum_t^T (A_{pt}h_t - B_{pt}f_t) + \frac{1}{D_{\mathrm{ext}}}\sum_k q_k G_{pk} \quad (9)$$

$$\left(\frac{1}{2\varepsilon}+\frac{1}{2}\right)h_p = \sum_t^T (C_{pt}h_t - D_{pt}f_t) + \frac{1}{D_{\mathrm{ext}}}\sum_k q_k \frac{\partial G_{pk}}{\partial n_0} \quad (10)$$

where T is the total number of discretized patches of the combined boundaries, while $2T$ represents the total unknowns of the system (i.e., f and h), and $\sum_k$ encompasses all the source charges of the system. The corresponding coefficient matrices are defined as follows:

$$A_{pt} = \int_{\Delta S_t}(G_{pt}-u_{pt})\mathrm{d}S, \quad B_{pt} = \int_{\Delta S_t}\left(\frac{1}{\varepsilon}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)\mathrm{d}S,$$
$$C_{pt} = \int_{\Delta S_t}\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\varepsilon}\frac{\partial u_{pt}}{\partial n_0}\right)\mathrm{d}S,$$
$$D_{pt} = \int_{\Delta S_t}\frac{1}{\varepsilon}\left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n}-\frac{\partial^2 u_{pt}}{\partial n_0 \partial n}\right)\mathrm{d}S, \quad (11)$$

where the integrations are performed on the small patch ΔS_t. To obtain the above form, f and h are assumed to be constant in each ΔS_t patch. When p and t are nearby patches, eq 11 is performed by direct integration, otherwise the kernel for each patch integral is taken as a constant. The patch properties such as the normal vector and area are determined by the discretization method. A "node patch" discretization is employed in this work and will be described in the following paragraphs. In a typical iterative solution of the linear system eqs 9 and 10, the matrix-vector multiplication (first summations in eqs 9 and 10) needs to be performed in every iteration step, which accounts for the major computational cost. However, these computations can be conveniently accelerated by using FMM: for all the local pairs of p and t as defined in the FMM oct-tree structure, direct integration is performed over the corresponding patches, while the far-field calculation is achieved through the multipole expansion approximation. In addition, FMM is also used in the summation over all the source point charges as appeared in the last terms in eqs 9 and 10.

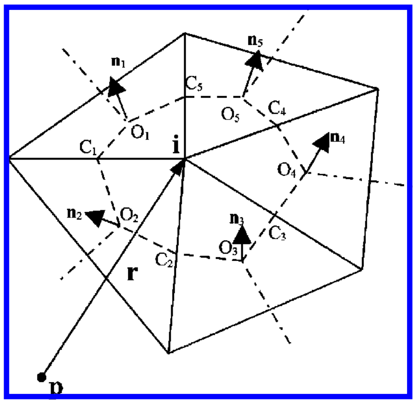

Figure 1. A "node patch" around the *i*th corner enclosed by the dashed lines is constructed on a triangular mesh. *O* and *n* are the centroid and normal vector of an element respectively, and *C* is the middle point of an edge.

There are two ways to treat the unknown f (or h) in the BEM approaches. The first is the so-called "constant element" approach which treats f (or h) as a constant on each element (face). Thus, the number of unknowns equals to the number of elements. Alternatively, f (or h) on each element can be obtained by linear interpolation from the unknowns of the three constituent nodes (also known as linear element approach), in which the number of unknowns equals the number of nodes. It is easy to show that, compared to the "constant element" approach, the "linear element" one leads to a reduction of the total number of unknowns by approximately a factor of 2, but a major disadvantage of the node-based approach lies in the introduction of additional complexity in its numerical implementation. In a recent communication, we introduced a node-patch scheme that appears to enjoy the benefits of both methods.[19]

The idea of the "node-patch" approach is to construct a "working" patch around each node instead of directly using the facet patch (element). We further assume that f (and h) is constant on this new "node-patch". A simple way to construct these new patches is illustrated in Figure 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 sets of points

$\{O_1, C_1, O_2, C_2, ...O_5, C_5, O_1\}$, where $\{O_l, l = 1, ..., 5\}$ are the centroids of the five adjacent triangles, and $\{C_l, l = 1, ..., 5\}$ are the midpoints of the five joint edges. It is easy to show that each triangular element contributes one-third of its area to the new "node-patch". Consequently, the far-field integrals on the new patch ΔS_i become

$$\int_{\Delta S_i} G_{pt} h_t dS \approx h_i G_{pi} \Delta S_i^a, \quad \Delta S_i^a = \frac{1}{3} \sum_{l \in \{L\}} \Delta S_l \qquad (12)$$

$$\int_{\Delta S_i} \frac{\partial G_{pt}}{\partial n} f_t dS \approx f_i G_{pi} \Delta S_i^b, \quad \Delta S_i^b = \frac{1}{3} \sum_{l \in \{L\}} \Delta S_l n_l \qquad (13)$$

where n_l is the unit normal vector of the lth neighboring element, ΔS_l is the area of the lth adjacent triangular element, all the neighboring elements of the ith node form a set $\{L\}$, and ΔS_i^b should be considered as a vector. For near-patch integration, a normal quadrature method is used as in the constant element method. Similar treatments apply to the integrations for the kernel u and its derivative, as well as for the second-order derivative terms if the dBIEs are used.

There are three main advantages of this "node-patch" approach in BEM. First, as aforementioned, 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 only additional computation is associated with the preprocessing of the geometric coefficients ΔS_i^a and ΔS_i^b in eqs 12 and 13. This, however, only constitutes a negligible portion of the total PBE solution time, and the geometric coefficients can also be saved for repeated use in iterative solving procedures. The second advantage, which is not so explicit, lies in the fact that relative to the linear element method, the "node-patch" method is significantly more efficient in searching and indexing the local list when used with any practical matrix storage format such as the Harwell−Boeing sparse matrix format or modified sparse column (row) format. Finally, in the "node-patch" method, the same as in the constant element method, the source and target are the same set of points, the nodes, which makes it straightforward to use any currently available FMM. Otherwise, if the source is different from the target as in the linear element method (where the convolution is done between two sets of data: the nodes and the quadrature points), still extra effort will be necessary to optimize the current FMM code to achieve comparable efficiency.

4. Krylov Subspace Methods

The discretized eqs 9 and 10 form a well-conditioned Fredholm second kind integral equation system, and a common practice for its efficient solution is to use Krylov subspace-based iterative methods. As the Fredholm second kind operator consists of an identity operator plus a compact operator whose eigenvalues only cluster at 0, it is well-known that the number of iterations in the Krylov subspace methods will be bounded, independent of the number of nodes in the discretization. Hence, the total number of operations required to solve eqs 9 and 10 is a constant (representing the number of iterations) times the amount of work required for a matrix vector multiplication. As will be discussed in next section, when the new version of fast multipole methods (FMM) are applied, the matrix vector product only requires $O(N)$ operations with an optimized prefactor; therefore, the linear equation system can be solved in asymptotically optimal $O(N)$ time.

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 $x_0 + \mathbf{K}_k$, where $\mathbf{K}_k = \text{span}\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\}$ and r_0 is the initial residual usually defined as $r_0 = b - Ax_0$. On the basis of different measures of the error and different types of matrices, there are many different implementations of the Krylov subspace method. As the matrix A in eqs 9 and 10 is nonsymmetric and there is no fast algorithm for multiplying the transpose of A with an arbitrary vector, in our solver, we have tested four different Krylov iterative subroutines from the open source package SPARSKIT developed by Saad and collaborators.[20,23] These are the full GMRES, the restarted GMRES, the biconjugate gradients stabilized (BiCGStab) method, and the transpose free QMR (TFQMR). Our preliminary numerical experiments show that all these solvers perform well in most cases, and not surprisingly, the full GMRES seems to perform the best. Also, as will be shown in Section 6, the number of iterations using the iterative solvers in SPARSKIT is often less than the numbers we observed in our previous implementations in which a different Krylov subspace package is used, partly because of our optimized initial guess and a few other improvements in the present code.

An interesting feature of the iterative solvers in SPARSKIT is the so-called "reverse communication protocol" (confer the ITSOL directory in SPARSKIT[20]), which avoids having to call 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 future input. Therefore, it is unnecessary to pass the parameters in the FMM subroutines to the Krylov solver, which means easier interface between the FMM and SPARSKIT, and easier memory management.

5. Adaptive Fast Multipole Methods

The fast multipole method was first invented by Greengard and Rokhlin in 1987[13] for the fast evaluation of the Coulomb interactions of N particles. For any given cluster of particles, as the far-field potential due to these particles is a smooth function and can be represented by a few terms of spherical harmonic expansions in 3D (the Laurent expansions in 2D), the interactions can therefore be efficiently accounted for using a divide-and-conquer strategy as follows: first, an oct-tree structure is generated so each childless box (leaf node) only contains a few particles; next, an upward sweep is executed to form the multipole expansions which carry the far-field information for all boxes, by using the particle information directly for the childless boxes, and by shifting the children's multipole expansions to parent level boxes (multipole-to-multipole); third, a downward sweep is used for each box to gather far-field information which is stored in a local expansion. At each level, the box first obtains very far-field information from its parent using a local-to-local

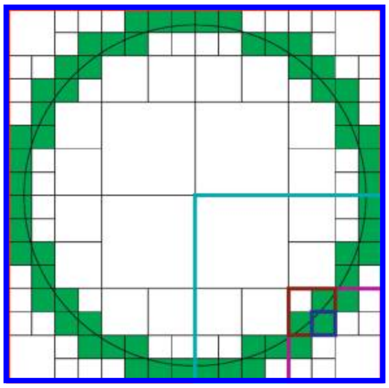

Figure 2. A schematic 2D adaptive tree structure.

translation and then shifts the multipole expansions from the "interaction list" (far-field boxes of itself which are not far field of its parent) to its local expansion (multipole-to-local); fourth, the local expansions of the childless boxes are evaluated at each particle location to account for all the far-field particle interactions; finally, the local particle interactions are evaluated directly. Notice that the number of boxes is $O(N)$ and the amount of work for local direction interactions is also $O(N)$; therefore, the algorithm is asymptotically optimal $O(N)$.

However, because of the large number of boxes in the interaction list and $O(p^2)$ operations for each multipole-to-local translation when p terms are used in the expansion, many numerical implementations reveal that the 1987 version of FMM is less competitive compared with other methods including the PME[9] and tree code,[12,24] and the prefactor in $O(N)$ is often >10 000. To further accelerate its performance, in 1997, a new version of FMM was presented by Greengard and Rokhlin for the Laplace equation,[14] 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. 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 Coulomb interactions. In our previous work,[6] 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 was the only LPB solver using the new version of FMM.

In this paper, we further improve our solver by using an adaptive new version of FMM. Unlike our previous implementation where a uniform oct-tree is generated, we remove those empty nodes in the oct-tree structure by only subdividing a box when the number of particles it contains is more than a prescribed number. Notice that this is important as all the unknowns are only located on the surface of the molecule. Figure 2 schematically shows a 2D adaptive tree structure for a circular boundary problem. There would be a total number of 256 smallest boxes when using 4 levels of box subdivisions in the uniform quad-tree structure, while only 64 boxes on the circular boundary are counted for in the adaptive tree structure. As shown by our preliminary numerical results in next section, the adaptive tree structure

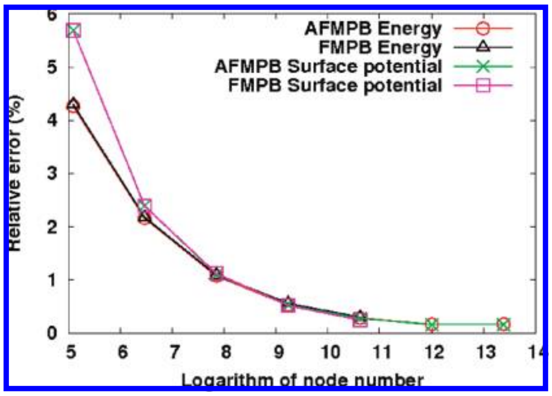

Figure 3. Accuracy of energy and potential calculations with the conventional and adaptive solvers. The relative errors of surface potentials are averaged over all node points.

not only improves the efficiency of the code but also reduces the required memory storage so larger problems can be computed.

Instead of discussing technical details of the adaptive new version of FMM, we refer the readers to existing literatures. The new version of FMM was introduced in ref 14 for the Laplace equation, the corresponding adaptive version was discussed in ref 16, the new version of FMM for the linearized PB equation (also called Yukawa or modified Helmholtz equation) was discussed in 17, and our LPB solver using a uniform oct-tree structure was presented in ref 6.

6. Benchmarks

A. A Spherical Cavity. The first system selected is a point charge located at the center of a spherical cavity. We examine the accuracy of the algorithm at different discretization resolution by comparing the calculated energy and potential to those from the analytical solution. We first note that for any relatively uniform particle distribution, AFMM and FMM maintain almost the same level of accuracy because the error is largely dependent on the box level and the number of terms used for expansion but not on how the data structure is represented, i.e., an adaptive vs nonadaptive tree structure. This is confirmed by the results displayed in Figure 3 that for both the energy and potential calculations the data lines obtained with AFMM overlap with those with FMM. It is also worth noting that the energy and potential calculations show similar errors, both of which reduce roughly linearly as the element size decreases. The energy and potential calculations converge with a relative error <0.2% when the surface mesh resolution is finer than 0.25 Å^2 (the surface area of a single triangular panel). At any given mesh resolution, the numerical error is bounded with the FMM (for far-field calculation) and the local numerical integration, but how the resolution and quality of meshes might affect the accuracy of the calculation is somewhat more difficult to quantify. While very good converging behavior (and well-defined converging resolution) is observed here for a spherical case, the calculations are performed on a single charge with a perfect geometry. Therefore, further studies would be necessary to assess the convergence criterion for more complex biological systems.

Table 1. Performance Comparison on a Spherical Cavity Case at Different Level of Discretization Resolution

number of elements	CPU (s)		memory (megabytes)		max. level	
	AFMM	FMM	AFMM	FMM	AFMM	FMM
162	0.05	0.13	2.7	2.7	3	2
642	0.21	0.62	7.0	7.9	4	3
2562	0.89	2.66	24.4	54.0	5	3
10242	4.63	11.44	113.3	241.0	6	4
40962	19.26	57.73	511.8	935.0	7	5
163842	78.35	-	2152.1	-	8	-
655362	1051.20	-	7900.7	-	9	-

A particular advantage of the adaptive algorithm, when compared to the nonadaptive one, is its lower memory usage. This is due to the fact that in a nonadaptive tree structure, when the elements only distribute on the surface as in BEM, a large number of boxes beyond the molecular surface are empty, leading to unnecessary memory usage for storing these empty boxes and their associated expansion coefficients. By contrast, the adaptive FMM continues to subdivide boxes only until the number of elements in a box has reached a predefined number, thus creating a practically 'uniform' partition of particles in all childless boxes regardless of their sizes. In our PB solver, the memory taken up by the FMM part constitutes a considerable part of total memory usage. But to what extent depends on how much information for local-field calculations the BEM saves during solution of the PBE. We here estimated the memory usage in a stand-alone FMM code for a test case where all the particles uniformly distribute on a spherical surface. Our results show a memory reduction of >10 fold with a 5-level-subdivision of the box, or more generally a reduction of $\sim 2^{n-1}$ fold when level n is greater than 6. In any real PB calculation, the above simplified analysis is not valid anymore because of several contributing factors, such as nonideal shape and/or charge distribution of the system, and additional memory usage by the other part of the program, but the overall trend remains evident that the adaptive algorithm still uses less memory than that of the nonadaptive one (Table 1). The comparison becomes even more favorable toward the adaptive solver when the subdivision becomes finer and/or the system size becomes larger. Because of this improvement, the PB calculations can now be performed on our desktop computer for systems with much more surface elements (e.g., 163 842 and 655 362 in Table 1) than what can be handled previously by the nonadaptive solver. For both solvers, the node-patch approach is used.

Furthermore, the adaptive FMM can strike a better load-balance for treating elements in the local list (calculated directly) and those in the far-field (calculated using expansion coefficients), while in the nonadaptive algorithm the partition between the local and far-field elements is greatly limited by the power growth of memory ($\sim 8^n$) as the number of levels n increases in an oct-tree data structure. As shown in Table 1, with 655 362 surface elements, the adaptive solver can handle a maximum tree level of 9 without causing any memory overflow problem on our 8-gigabyte desktop computer. However, for the nonadaptive algorithm, the maximum level that can be handled is only 6 (data not shown). Further tests reveal that level 9 enables the most balanced local and far-field calculations, thus is optimal for this particular case. When the calculation is otherwise performed at level 6, the FMM calculation is very unbalanced in a sense that too many elements are assigned to the local list for direct calculation. Specifically, the direct calculation takes more than 90% of the total computing time, while the far-field part takes less than 5%. The poor load-balance significantly compromises the overall efficiency of the calculation. For most of the calculations, an average speedup of 2−3 fold has been observed by using the adaptive algorithm, while better performance is generally expected for larger systems. Therefore, as compared to the nonadaptive solver, our new adaptive implementation not only makes more efficient use of the memory but also increases the calculation speed quite significantly.

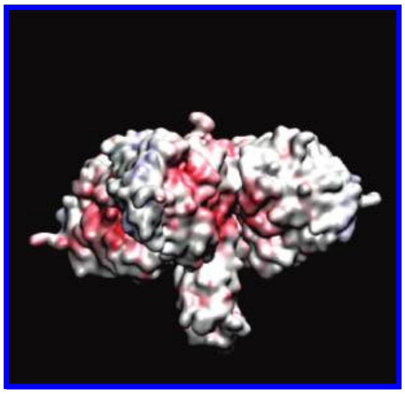
Figure 4. Electrostatic potential surface of the acetylcholinestase.

Table 2. Performance Comparison on the Acetylcholinesterase Tetramer

	old FMPB	AFMPB
solvation energy (kcal/mol)	−8341.3	−8342.4
CPU time (s)	695.5	94.2
memory (gigabytes)	1.40	1.05
max. level	6	9
number of iteration	18	15

B. Acetylcholinesterase Tetramer. As a representative protein system, we chose the acetylcholinesterase tetramer, which contains 36 638 atoms with a dimension of 135 Å × 112 Å × 115 Å (Figure 4). The molecular surface (also known as the solvent-excluded surface), which is the surface traced by the inward-facing surface of the probe sphere, is used as the boundary. The surface discretization using MSMS[26] resulted in 124 254 triangular elements and 62 095 nodes. Both the adaptive and nonadaptive solvers can handle this system well on a typical 2008 desktop computer, giving very comparable solvation energy (see Table 2). It would have seemed surprising at the first sight that the adaptive solver uses almost the same amount of memory as the nonadaptive one, but when you note that far more levels of box subdivision are used by the adaptive solver than the nonadaptive one, the seemingly conflicting results are actually easy to understand. Because of its ability to involve more tree levels, the new adaptive solver runs about 7 times faster than the old one, one of the greatest speedups observed in all our test calculations. The main reason for the observed

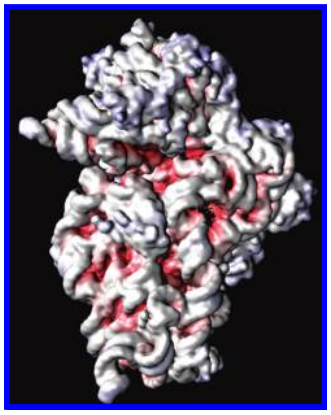

Figure 5. Electrostatic potential surface of the 30S ribosome subunit.

acceleration as aforementioned is because the new solver can have a more balanced work load for both the local and far-field computation. By contrast, the old nonadaptive solver cannot run with more than 6 levels, thus leading to a too heavy local computation load. Another acceleration factor comes from the reduction of the iterative steps (Table 2) in solving the linear system by using SPARSKIT as analyzed above.

C. 30S Ribosome Subunit. Finally, we show the results of a 30S ribosome system (PDB code: 1fjf[25]), which is nontrivial to compute with other or our earlier PB solvers using a serial version on a desktop, because of its far greater memory requirement. The 30S ribosome subunit consists of 21 peptides and a 1540-nucleotide RNA subunit, with a total of 88 431 atoms (including hydrogen atoms) and a dimension of 211 Å × 177 Å × 200 Å (Figure 5). Because of its size, the software MSMS fails to generate a molecular surface mesh for ribosome. So, the Gaussian surface, which is a level-set of the summation of the spherically symmetrical Gaussian density distribution centered at each atom of the molecular, is used as the molecular boundary, and the surface discretization is performed using the software Gamer.[27] The surface discretization results in 343 028 triangular elements and 171 288 nodes. The edge length resolution is about 1 Å. The whole computation takes ~21 min on our desktop machine (Intel(R) Xeon(TM) CPU 3.00 GHz, 4GB memory), with a memory usage of 2.6 GB. An 11-level tree structure is used for optimal efficiency, and 92 iteration steps are taken to obtain a converged solution. Figure 5 shows the computed electrostatic potentials mapped on the molecular surfaces of the 30S ribosome.

7. Conclusions

We have described a new implementation of our BIE-based PB solver that uses an adaptive FMM for accelerating the *N*-body type surface integration. Other salient points of our current implementation include (1) the well-conditioned formulation that is extended to multidomain systems, (2) the development of an efficient patch-node scheme for surface discretization, and (3) efficient use of Krylov subspace method for iterative solution of the linear system. The adaptive FMM reduces the total number of boxes by using nonuniform oct-tree structure and thus leads to significant reduction in memory usage. Because of its ability to keep a better load-balance for the local and far-field calculations, the speedup is also quite significant when compared to our earlier version of the nonadaptive solver.

The resulting solver was tested with several applications. The accuracy of the new algorithm was first examined by direct comparison with the analytical solution of a point charge located at the center of a spherical cavity. It is found that the solvation energy of our spherical cavity with radius 50 Å converges with a relative error <0.2% when the surface mesh resolution is below 0.25 Å^2 of each triangular element. Our new PB solver significantly outperforms our earlier nonadaptive solver and shows a stronger linear growth of both memory and computational cost with the number of unknowns. Primarily because of more efficient memory allocation, the new solver enables very large-scale calculation to be executed on a typical 2008 desktop machine. A PB calculation on the 30S ribosome (88 431 atoms) illustrated the capability of the code. The new solver is also very suitable for doing calculation on large-scale biomolecular assembly or complex systems that comprise a set of molecules with large separations between them, though we do not show any test calculation for such cases. Further applications of the methodology are in progress, including the coupling with molecular dynamics/Brownian dynamics simulation and other continuum models for studying molecular interactions and dynamics of biological systems.

However, in order to perform dynamics simulation or study other electrostatics-controlled dynamical process, our code needs further improvement. To do this, our current efforts include (1) generating an optimized oct-tree structure using spectral graph theory,[21] and (2) parallel implementation of the code on computers with shared and/or distributed memory. Another important direction is to come up with a more efficient way to generate molecular surfaces, which seems to be the current bottleneck for performing a fully dynamical simulation (PB solution at every time step). Finally, the surface specification itself is an important and open issue as aforementioned. As the general practice in BEM, this work uses a single surface separating the solute and the solvent. On the other hand, in many finite-difference methods, a second surface, the so-called Stern layer, is introduced to account for the fact there is a layer in the solvent to which mobile ions cannot access. Complicated by some other factors (parameters) in the setup of a PB calculation that can also affect the final results, it is hard to conclude thus far which surface specification is the best. Likewise, the surface model adopted in BEM will need further tests and comparisons with experiments or other more accurate computations.

Acknowledgment. We 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,[20] MSMS,[26] and Gamer[27] for mesh generation, VMD[28] for visualization, and several important subroutines in the new version of FMM from Profs. Greengard and Rokhlin's group. This work was supported by HHMI, NIH, NSF (J.H.: NSF0811130 and NSF0411920, J.A.M.: MCB0506593), and the NSF Center of Theoretical Biological Physics (CTBP). B.Z. is partially funded by the "100 Talents Projects" of the Chinese Academy of Sciences, China. X.C. is partially funded by the Computer Science and Mathematics Division at Oak Ridge National Laboratory. Their support is thankfully acknowledged.

References

(1) Baker, N. A. *Methods Enzymol.* **2004**, *383*, 94.

(2) Baker, N. A.; Sept, D.; Joseph, S.; Holst, M. J.; McCammon, J. A. *Proc. Natl. Acad. Sci. U.S.A.* **2001**, *98*, 10037.

(3) Holst, M.; Baker, N. A.; Wang, F. *J. Comput. Chem.* **2000**, *21*, 1319.

(4) Boschitsch, A. H.; Fenley, M. O.; Zhou, H. X. *J. Phys. Chem. B* **2002**, *106*, 2741.

(5) Kuo, S.; Altman, M.; Bardhan, J.; Tidor, B.; White, J. Fast methods for simulation of biomolecules electrostatics. Proceedings of the IEEE/ACM International Conference on Computer Aided Design; San Jose, CA, November 10–14, 2002, p 466.

(6) Lu, B.; Cheng, X.; Huang, J.; McCammon, J. A. *Proc. Natl. Acad. Sci. U.S.A.* **2006**, *103* (51), 19314.

(7) Phillips, J. R.; White, J. A Precorrected-FFT Method for Capacitance Extraction of Complicated 3-D Structures. International Conference on Computer-Aided Design; San Jose, CA, November 6–10, 1994.

(8) White, J.; Phillips, J. R.; Korsmeyer, T. Comparing Precorrected-FFT and Fast Multipole Algorithms for Solving Three-dimensional Potential Integral Equations. Proceedings of the Colorado Conference on Iterative Methods; Breckenridge, CO, April 4–10, 1994.

(9) Darden, T.; York, D.; Pedersen, L. *J. Chem. Phys.* **1993**, *98* (12), 10089.

(10) Lee, H.; Darden, T.; Pedersen, L. *Chem. Phys. Lett.* **1995**, *243* (3−4), 229.

(11) Abramowitz, M.; Stegun, I. A. *Handbook of Mathematical Functions*; Dover, New York, 1970.

(12) Appel, A. W. *SIAM J. Sci. Stat. Comput.* **1985**, (6), 85.

(13) Greengard, L.; Rokhlin, V. *J. Comp. Phys.* **1987**, *73* (2), 325.

(14) Greengard, L.; Rokhlin, V. *Acta Numer.* **1997**, *6*, 229.

(15) Greengard, L.; Wandzura, S. *IEEE Comput. Sci. Eng.* **1998**, *5* (3), 16.

(16) Cheng, H.; Greengard, L.; Rokhlin, V. *J. Comp. Phys.* **1999**, *155* (2), 468.

(17) Greengard, L.; Huang, J. F. *J. Comp. Phys.* **2002**, *180* (2), 642.

(18) Fast Multipole Methods (Beta). http://www.fastmultipole.org/ (accessed Dec 14, 2008).

(19) Lu, B.; McCammon, J. A. *J. Chem. Theory Comput.* **2007**, *3*, 1134.

(20) SPARSKIT. http://www-users.cs.umn.edu/∼saad/software/SPARSKIT/sparskit.html (accessed Mar 8, 2005).

(21) Chung, F. R. K. Spectral Graph Theory; Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, RI, 1997.

(22) Dong, F.; Vijayakumar, M.; Zhou, H. X. *Biophys. J.* **2003**, *85* (1), 49.

(23) Saad, Y.; Schultz, M. *SIAM J. Sci. Statist. Comput.* **1986**, *7*, 856.

(24) Barnes, J.; Hut, P. *Nature* **1986**, *324*, 446.

(25) Wimberly, B. T.; Brodersen, D. E.; Clemons Jr., W. M.; Morgan-Warren, R.; Carter, A. P.; Vonrhein, C.; Hartsch, T.; Ramakrishnan, V. *Nature* **2000**, *407*, 327.

(26) MSMS. http://www.scripps.edu/∼sanner/html/msms_home.html (accessed Feb 5, 1996).

(27) Yu, Z.; Holst, M.; Cheng, Y.; McCammon, J. A. *J. Mol. Graph. Model.* **2008**, *26*, 1370.

(28) VMD. http://www.ks.uiuc.edu/Research/vmd/ (accessed Mar 20, 2006).

CT900083K