Accelerated Poisson–Boltzmann Calculations for Static and Dynamic Systems

RAY LUO,[1] LAURENT DAVID,[2] MICHAEL K. GILSON[3]
[1]*Department of Molecular Biology and Biochemistry, Department of Information and Computer Science, University of California, Irvine, California 92697-3900*
[2]*AstraZeneca R&D Lund, S-221 87 Lund, Sweden*
[3]*Center for Advanced Research in Biotechnology, University of Maryland Biotechnology Institute, Rockville, Maryland 20850*

Received 12 November 2001; Accepted 11 April 2002

Abstract: We report here an efficient implementation of the finite difference Poisson–Boltzmann solvent model based on the Modified Incomplete Cholsky Conjugate Gradient algorithm, which gives rather impressive performance for both static and dynamic systems. This is achieved by implementing the algorithm with Eisenstat's two optimizations, utilizing the electrostatic update in simulations, and applying prudent approximations, including: relaxing the convergence criterion, not updating Poisson–Boltzmann-related forces every step, and using electrostatic focusing. It is also possible to markedly accelerate the supporting routines that are used to set up the calculations and to obtain energies and forces. The resulting finite difference Poisson–Boltzmann method delivers efficiency comparable to the distance-dependent dielectric model for a system tested, HIV Protease, making it a strong candidate for solution-phase molecular dynamics simulations. Further, the finite difference method includes all intrasolute electrostatic interactions, whereas the distance dependent dielectric calculations use a 15-Å cutoff. The speed of our numerical finite difference method is comparable to that of the pair-wise Generalized Born approximation to the Poisson–Boltzmann method.

© 2002 Wiley Periodicals, Inc.   J Comput Chem 23: 1244–1253, 2002

Key words: finite-difference Poisson–Boltzmann; modified incomplete Cholsky conjugate gradient; molecular dynamics

Introduction

A common problem in computational chemistry is that atomic level models for solution processes can be simulated for only a few nanoseconds on a routine basis, while biochemical processes, such as protein folding, are often on the time scale of microseconds or longer. Much simpler models based on lattice[1–4] or off-lattice[5] representations with residue-level resolution have been developed to study such processes. However, an atomic-detail description of biomolecules is often important in elucidating their structures and functions. Therefore, a balance must be achieved in developing models with simple interactions that permit fast calculations without the loss of the important atomic detail of biomolecules.

A class of continuum solvent models based on the classical electrostatic theory have been developed for this purpose.[6,7] In these models, the solute is represented by an atomic detail model as in a molecular mechanics force field, while the solvent molecules are treated as a structureless continuum. The solute intramolecular interactions are computed by the usual force field terms, while the solute–solvent and solvent–solvent interactions are computed by a classical electrostatic model. The electrostatic model represents the solute as a dielectric body whose shape is defined by atomic coordinates and radii.[8–10] The solute contains a set of charges that produce an electrostatic field in the solute region, and in the solvent region represented by the dielectric continuum. The electrostatic fields in such a system may be computed by solving the Poisson equation.[11–13] A dissolved electrolyte may be accommodated by use of the Poisson–Boltzmann equation instead of the Poisson equation.[14] This class of methods has been demonstrated to be reliable in reproducing the energetics and conformations compared with explicit solvent simulations and experimental measurements for a wide-range of systems.[15–19]

Although these are physically simple and clean models, the numerical solution of the Poisson–Boltzmann (PB) equation has been a bottleneck, largely limiting their application to calculations with static structures only. Typical methods which involve discretization of the partial differential equation into a system of linear equations that tends to be rather large: it is not uncommon

Correspondence to: M. K. Gilson; e-mail: gilson@umbi.und.edu to have a million unknowns in biochemical applications. In addition, the setup of the linear system before the numerical solution and postprocessing to obtain energies and forces are both nontrivial. The most commonly used finite-difference (FD) approach was first introduced into biochemical studies in the early 1980s.[11,12,20,21] The programs Delphi,[21] UHBD,[22] and Grasp[23] have greatly assisted its adoption by the biochemistry community. Boundary element approaches also have been widely used, especially in the field of quantum chemical applications for small organic molecules.[24] They have also been used for molecular mechanics,[25–27] although less widely than the finite difference approach. One advantage in applying boundary element approaches in biochemistry lies in the fact that the dimension of the linear system is much smaller than that from the finite difference scheme due to the different dimensionality (3D vs. 2D). However, the corresponding linear system is much denser, making it more difficult to solve. The finite-element approach was also introduced into chemistry, with the idea that it could reduce the number of grids when compared with the uniform finite difference grid with the same grid resolution near solute atoms.[28–30] However, the regular pattern in the matrix from finite difference discretization is lost, making it harder to write a highly efficient solver. Recently, multigrid approaches have been introduced into the field of biochemistry to speed up the convergence of various solvers.[27–31] Their performance gain is especially useful for very large systems.[32]

Even with the above progress in the numerical solution of the PB equation, it has remained impractical if not impossible to apply a PB solvent to simulations when intensive conformational sampling is needed, as in molecular dynamics (MD) and Monte Carlo techniques. The earliest attempts to use PB in dynamic simulations date back to as early as the 1990s when Davis et al.,[33] Zauhar,[34] Sharp,[35] Luty et al.,[36] Neidermeier et al.,[37] and Gilson et al.[38,39] contributed to adapting PB for dynamic simulations. However, these applications were sharply limited in scope because the cost of using PB on macromolecules was prohibitively high. In fact, the per-step simulation cost is higher with the finite difference approach than that with explicit water even at 1-Å grid spacing, although it can be argued that the solvent is always equilibrated in PB, whereas it takes a long time to achieve equilibration in explicit water simulations. Its inefficiency sharply limits practical usage of the PB model in routine MD simulations of macromolecules. Recently, there has been renewed interest in finding ways to apply PB in dynamic simulations.[40–42] Still, inefficiency remains an obstacle to its application to macromolecules of biochemical interest, despite recent efforts to make the numerical solution of the PB equation more efficient and more accurate as mentioned above. As will be shown later, this is not a surprise because solving the linear system is only a portion of the computational cost in applying PB to dynamic simulations.

Due to the computational expense of solving the PB equation numerically, considerable effort has been invested in approximating the solution of the PB equation, via methods such as the Generalized Born (GB) model,[43] Induced Multipole Solvent model,[44,45] the Dielectric Screening model,[46,47] and others. The pairwise GB model, in particular, has been widely accepted as an efficient estimation of the solution of the PB equation as recently reviewed.[48,49] Many variants of the original GB model exist.[50–57] However, none of these approximate models can treat the detailed solvation of biomolecules as accurately as numerical solutions of the PB equation. This is especially so for the desolvation of charged groups, which occurs quite often in protein dynamics. Thus, our recent publication comparing GB and PB for the active site of HIV protease shows that the GB model gives similar dynamics but significantly different energetics.[58,59] Also, inclusion of salt effects in the generalized Born model remains challenging.[60]

Successful application of any Poisson–Boltzmann solvent model (including GB) in molecular mechanics requires careful parameterization of the atomic cavity radii used in defining the dielectric boundary,[61,62] as well as a set of compatible surface tension coefficients for the nonpolar solvation energy (i.e., the SA term).[63,64] We will delay such investigations to later publications. Thus, no effort is made here to study the quality of the PB/SA solvent model in its application to static or dynamic systems. Our sole focus here is on the efficiency of the numerical method. The Methods section of this article describes techniques for accelerating the finite difference solution of the Poisson–Boltzmann equation. These include improving the core iteration, the assignment of boundary conditions, and the calculation of energies and forces. The Results section describes the performance of these methods by comparing with existing approaches that are known to be accurate.[38,39] Finally, future directions in applying PB to biochemical systems are discussed.

Methods

The current article focuses on the finite-difference solution of the Poisson–Boltzmann equation. Many excellent articles and reviews have outlined this rather straight-forward algorithm.[12,15,65] Solving the PB equation involves the following initial steps: mapping atomic charges to the finite difference grid points;[13,66] determining the boundary between high-dielectric (i.e., water) and low-dielectric (i.e., solute interior) regions;[13,67] and assigning boundary conditions, i.e., the electrostatic potentials on the boundary surfaces of the finite-difference grid. These steps allow the partial differential equation to be converted into a linear system $\mathbf{Ax} = \mathbf{b}$ with the electrostatic potential on every grid point, $\mathbf{x}$ as unknowns, the charge distribution on the grid points as the source $\mathbf{b}$, and the dielectric constant and salt-related terms wrapped into the coefficient matrix $\mathbf{A}$, which is a seven-banded symmetric matrix.[12] Standard numerical methods can be used to solve the linear system.[68] Methods used in chemistry include Gauss–Seidel,[13] SOR,[21] preconditioned conjugate gradient,[20] and others.[68,69] Once the linear system is solved, the solution is used to compute the energies and forces. In the following, we discuss methods used to improve the efficiency of these steps in applying PB to molecular mechanics calculations.

Acceleration of the FDPB Core Iteration Routine

This subsection describes the methods used to accelerate the FDPB core iteration routine that solves the linear system. Many solvers are described in the literature.[68,69] We focus on a class of preconditioned conjugate-gradient methods, partly because they yield good numerical behavior in dynamic simulations, and partly because they are easy to implement. These algorithms can be summarized in the following pseudocode:[68]

```
$x_0$ = initial guess;
$r_0$ = b − A $x_0$;
solve M $r'_0$ = $r_0$;
$p_0$ = $r'_0$;
for i = 0, 1, 2, . . . until converged
    $a_i$ = $\langle r_i, r'_i\rangle / \langle p_i, Ap_i\rangle$;
    $x_{i+1}$ = $x_i$ + $a_i$ $p_i$;
    $r_{i+1}$ = $r_i$ − $a_i$ $Ap_i$;
    if converged then exit;
    solve M $r'_{i+1}$ = $r_{i+1}$;
    $b_i$ = $\langle r_{i+1}, r'_{i+1}\rangle / \langle r_i, r'_i\rangle$;
    $p_{i+1}$ = $r'_{i+1}$ + $b_i$ $p_i$;
end for
```

where **M** is a preconditioning matrix chosen to be close to **A, r** and **p** are auxiliary vectors, and $\langle x, y\rangle$ represents the inner product of x and y.

Here, we focus on an efficient conjugate-gradient solver, Modified Incomplete Cholsky Conjugate Gradient (MICCG).[70] We then further explore approximations that maximize computational efficiency. These include choosing a reasonable convergence criterion, exploiting the small perturbations in molecular mechanics simulations to set up a good initial guess for the electrostatic potentials, computing FDPB energies/forces less frequently than other force field terms during molecular mechanics simulations, and using electrostatic focusing.

Speedup from Modified Incomplete Cholsky Conjugate Gradient

MICCG vs. ICCG: MICCG is an extension of the Incomplete Cholsky Factorization (ICCG)[71] to precondition the conjugate gradient solver. The latter approach in the context of chemical and biochemical applications was discussed by Davis and McCammon.[20] Here we do not plan to go over the technical details that have been described in many textbooks.[68,69] Both methods construct the preconditioning matrix (preconditioner) as $\mathbf{M} = (\mathbf{L} + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})$,[71] where **L** is the strictly lower triangle of **A** (i.e., without the diagonal elements of **A**), and **U** is the strictly upper triangle of **A.** The only difference between the two is the construction of the diagonal of the preconditioner **D.**[70,72] In MICCG:

$$
\begin{aligned}
d_i^{-1} = A_{i,1} &- A^2_{i-1,2}(A^2_{i-1,2} + \alpha A^2_{i-1,3} + \alpha A^2_{i-1,4})d_{i-1} \\
&- A^2_{i-nx,3}(\alpha A^2_{i-nx,2} + A^2_{i-nx,3} + \alpha A^2_{i-nx,4})d_{i-nx} \\
&- A^2_{i-nxny,4}(\alpha A^2_{i-nxny,2} + \alpha A^2_{i-nyny,3} + A^2_{i-nxny,4})d_{i-nxny} \qquad (1)
\end{aligned}
$$

where the elements of the seven-banded matrix **A** of dimension $n_x n_y n_z$ are denoted by $A_{i,1}$, the diagonal elements at row i; $A_{i,2}$, $A_{i,3}$, and $A_{i,4}$ are the three nonzero off-diagonal elements at row i. It was found that construction of **M** this way often makes the algorithm converge faster than ICCG for model problems.[72] Note that the above expression goes back to that of ICCG when $\alpha = 0$. Application of MICCG in small organic systems with the default α value (0.95) was first reported by Holst.[31] The influence of α on its performance on macromolecular systems is unclear.

Efficient Implementation of MICCG: given the preconditioner $\mathbf{M} = (\mathbf{L} + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})$ in MICCG, the solution of $\mathbf{M}r' = r$ in the preconditioned conjugate gradient algorithm (see above) requires forward substitution, backward substitution, and division of a diagonal matrix.[68] This is in addition to the matrix vector multiplication, two inner products, and three vector updates in the original conjugate gradient algorithm.[68] There are many ways to implement the MICCG method, but their efficiencies are different.[68] Eisenstat suggested two approaches to reduce the work load per iteration as summarized below.[73]

(1) Scaling of the diagonal matrix to unity: by eliminating the diagonal matrix **D** in **M,** $n_x n_y n_z$ divisions per iteration can be removed. In addition to the reduced instruction count per iteration, the removal of divisions also makes the instruction pipeline flow smoother in a modern pipelined CPU.[73]

(2) Direct preconditioning: Eisenstat's second suggestion further reduces the instruction count per iteration by eliminating the extra solution of $\mathbf{M}r' = r$ from the preconditioned conjugate gradient method so that the preconditioned algorithm has a work load very similar to the unconditioned conjugate gradient method, i.e., only one matrix vector multiplication is needed in addition to the two inner products and three vector updates. This is achieved by explicitly preconditioning the linear system.[73] However, the matrix vector multiplication becomes much more complex in the explicitly preconditioned version of the algorithm.[73]

As shown in the Results section, the FDPB core iteration routine using MICCG improves speed, but not enough for many applications to large macromolecules, such as interactive display of electrostatic potentials and molecular dynamics simulations. However, there are many ways to further speed up the calculation when approximations are allowed. In the following, we study the accuracy and efficiency of these approximations.

Adjustment of the Convergence Criteria

The convergence criterion directly influences the number of iterations needed in the conjugate gradient algorithm. Ideally, we would require an algorithm to stop only when it reaches zero error, i.e., zero residual $\|\mathbf{r}\|$. However, it is almost impossible to reach zero within reasonable computer time for typical biochemical systems. It is thus a common practice to use relative residual, $\|\mathbf{r}_i\|/\|\mathbf{r}_0\| \leq 10^{-6}$ for FDPB calculations in these applications.[20] Here we study the effect of convergence criterion on electrostatic energies and forces in molecular mechanics simulations, and on visualization of the electrostatic field for molecules of biological interest. With the common practice 10^{-6} as a "standard," we have studied several relaxed convergence criteria, 10^{-3}, 10^{-2}, and 10^{-1} on efficiency and accuracy.

Speedup in Molecular Dynamics Simulations

A good initial guess of the potential distribution can speed up the convergence of a FDPB solver. However, a typical scenario is that the better the guess, the more effort must be invested in making the guess. Otherwise, there is no need to solve the PB equation by the FDPB solver. Fortunately, without much effort we can obtain a very good guess for the potential in the context of molecular dynamics or energy minimization. This may be shown by studying the convergence behavior of the FDPB core routine in those calculations. It is found that the spatial distribution of electrostatic potential changes little due to the small perturbation in dielectric and charge distributions at each step during such simulations. Therefore, an MICCG run converges significantly faster when it uses the potential distribution from the previous step as input. Here, we term an FDPB calculation that uses the potential from a recent prior calculation as an update run. In contrast, the first MICCG run is termed a *de novo* run, i.e., a calculation without any memory. Typically, the guessed potential is set to zero everywhere for a *de novo* run.

To study the convergence behavior of the FDPB core routine in a molecular dynamics simulation, we tested the update runs with the MICCG code on a trajectory of HIV Protease with a time step of 1 fs. The numbers of MICCG iterations to reach convergence were collected for update periods from 1 fs to 1 ps.

It should be emphasized that careful bookkeeping is required to best utilize the small perturbation in potential distribution. In particular, the finite-difference grid must be fixed in the laboratory frame while the solute molecule is moving during a simulation.

Intermittent Updating of FDPB Forces in Simulations

The small changes in structure from step to step in molecular mechanics simulations suggest that it may not be necessary to compute FDPB energies and forces at every step during a simulation. Indeed, it is a common practice to update FDPB forces, say, every 10 MD steps or more instead of every step.[37,42] This effectively speeds up the PB portion by a factor of 10 or more when properly used. However, the accuracy of this approximation has not been well documented. In this work, we systematically study the relation of the error to the update period. This is done by computing the RMS relative deviation of all nonzero electrostatic forces of HIV Protease with different update periods, for the same trajectory as in the previous subsection. Further, two different grid spacings are used to examine the influence of the spacing on this approximation.

Electrostatic Focusing

Electrostatic focusing has been widely used for FDPB calculations on proteins.[13] Often it is used to study a local interaction within a protein, such as its active site or a salt bridge. In such applications, long-range interactions with the rest of the protein can be accurately represented by the previous coarse grid calculation. Also, it is used to balance the quality of boundary conditions, which require the grid boundary to be far away from the surface of the solute, and the quality of FDPB calculations, which require the grid spacing to be as fine as possible. A fine grid often results in an unmanageably large grid dimension if high-quality boundary conditions are used.

In this study, we study the efficiency and accuracy of this technique in the context of molecular dynamic simulations. We again use the HIV Protease, a protein of about 1800 atoms as our test system. The electrostatic energies and forces from a grid of 100 Å^3 and 1 Å grid spacing is used as a standard in the comparison. In this run, the protein only occupies at most 50% of the grid dimension, guaranteeing high-quality boundary conditions. In the FDPB with focusing, we use a grid of 50^3 and 2-Å grid spacing for the first pass, and a grid of $81 \times 61 \times 51$ and 1-Å grid spacing for the final pass. The final grid is large enough to enclose the protein with over 10 grids buffer zone between the protein surface and the boundary.

Table 1. Profiling Analysis of 10 Independent *de novo* FDPB Runs for HIV Protease with UHBD.

Routine	CPU usage(s)	Percent of total
Core iteration	807.12	57.9
Boundary conditions	294.24	21.0
Energies and forces	283.44	20.2
Dielectric boundary	5.82	0.4
Others	7.01	0.5

Convergence criterion: 10^{-3}. Grid dimensions: $101 \times 101 \times 101$. The boundary conditions are the sum of potentials from individual atoms as independent Debye Huckel spheres.

Supporting Routines

This section discusses acceleration of the supporting routines for the FDPB continuum solvent. We start by analyzing the CPU usage of the FDPB module in UHBD[22] with a profiling analysis on an SGI Octane workstation. The results, shown in Table 1, indicate that supporting routines can take almost as much time as the core iteration routine. Note that here, the van der Waals surface is used to define the dielectric boundary, rather than the more time-consuming Richards surface.[74] In the following, we first show how to improve the performance of the two most expensive supporting routines, those used to set up the boundary conditions and to compute energies and forces. Then we discuss issues related to setup of the dielectric boundary.

Setup of Boundary Conditions

The boundary condition routine computes the electrostatic potential at every grid point on the six boundary surfaces of the finite difference grid. For a grid of $101 \times 101 \times 101$, this needs about 60,000 potential calculations, each of them using all atoms of the solute molecule if high-quality boundary conditions (sum of potential from individual atoms) are needed. For a typical protein of hundreds to thousands of atoms, this is too costly computationally. Here we make two approximations in the boundary potential calculation. First, residues instead of atoms are used in the potential calculation, where each residue is represented by a point dipole and point monopole if any. Second, the boundary potentials are computed on a very coarse boundary grid; the spacing is four times as coarse as the spacing for the finite difference grid. After the boundary potentials are computed on the coarse grid, the potentials on the remaining boundary grid points are obtained by the trilinear interpolation method.[66] The performance of these two approximations will be discussed in the Results section.

Computation of Energy and Forces

The electrostatic energy from a FDPB calculation may be computed as $1/2\,\sum_i q_i\phi_i$, where q_i is the charge of atom i and ϕ_i is the potential at atom i as interpolated from the finite difference grid. This energy includes not only the electrostatic solvation energy (i.e., reaction field energy), but also the self-energy (which exists even in the absence of solvent and any other charges) and the finite-difference Coulombic energy, which results from the interaction of each charge with each other charge as if no solvent were present.[10] The self-energy is a pure artifact of the finite-difference approach, and must be removed. The finite-difference Coulombic energy is an accurate approximation of Coulomb's law at long range, but not at short range.[36] Also, the the finite-difference Coulombic energy is often not needed, such as when one is only interested in the solvation energy. As a consequence, the results of an FDPB calculation must be further processed to compute the reaction field energy and forces. Two approaches exist. The Delphi program computes the apparent induced surface charges to obtain the reaction field energy. The reaction field forces can be computed from the these surface charges by an approach adopted in Friesner's group.[42] If only reaction field interactions are needed, the Delphi approach can be used to take advantage of its well optimized code. Alternatively, the UHBD FDPB-related energy/force routines implement Luty and McCammon's analytical correction[36] to delete the self- and finite-difference Coulombic energies.

We have chosen the second method because it can be used to compute both long-range Coulombic and reaction field interactions in FDPB, while short-range Coulombic interactions can be computed along with the van der Waals interactions as in the usual molecular mechanics simulations. This approach is useful because it accounts for long-range electrostatic interactions with no distance cutoff at minimal extra computational cost. The accuracy of the finite-difference Coulombic interaction of a pair of charges depends upon the distance between the charges and upon the grid spacing; it has been shown that a distance of six grid units is enough to yield accurate results.[36] Here, we use 8 Å (eight grids) as the cutoff to ensure accurate results and because all 1–2, 1–3, and 1–4 interactions are within this distance. The 1–2 and 1–3 interactions are removed entirely, and the 1–4 interactions are removed and added back with their usual scaling factor to implement the force-field correctly. Furthermore, we compute the van der Waals interactions with the same 8-Å cutoff, so that this avoids implementing multiple cutoffs for nonbonded interactions. The efficiency of the new approach is described in the Results section.

Setup of Dielectric Map

The dielectric boundary in FDPB calculations is based upon a definition of the molecular surface, and it is a common practice to use the Richards molecular surface[74] for this purpose. However, setting up this surface can be time consuming, making for a third bottleneck in the FDPB method. Here, we use the van der Waals surface, which represents the low-dielectric molecular interior as the union of the atomic van der Waals volumes. This region is generated efficiently by a single loop over the atom list, so the method is very fast. As a consequence, the surface calculation does not contribute significantly to the CPU usage listed in Table 1. Speeding the calculation of a Richards-like molecular surface is itself an important and nontrivial issue that has been addressed by a number of groups in the recent years.[41,52,75,76] We delay the investigation of this issue to a later publication.

Efficiency of the Resulting FDPB Method for Dynamics Simulations

This section describes the implementation of the optimized FDPB method in the context of dynamics simulations. We use two simulations of HIV Protease to evaluate the efficiency of the new method: one simulation with the new FDPB solvation method, the other with a distance-dependent dielectric constant (DDD) solvation method as a timing reference. The details of both simulations are now given, followed by a description of the molecular model used in the simulations.

Simulation with the FDPB Method

All simulations used a stochastic dynamics algorithm[77] to thermalize the systems. The time step was 1 fs, the friction coefficient, γ, was set to 10 ps^{-1},[78] and all CHARMM covalent energy terms were included. In the FDPB method, the dielectric constants of internal and external regions are set to 1 and 80, respectively. The ionic strength is zero. An atomic cutoff distance of 8 Å was used for the short-range pair-wise Coulombic interactions and van der Waals interactions. The nonbonded list was updated every step using a secondary nonbonded list of 12-Å cutoff distance. The secondary nonbonded list was updated every 1000 steps. The long-range Coulombic and solvent reaction field interactions were obtained from the FDPB calculations. The convergence criterion for FDPB iteration was 10^{-2}, and a two-level focusing technique was used for the FDPB calculations, as described above. The coarse grid had a spacing of 2 Å and dimensions of $49 \times 33 \times 27$ grid units; the fine grid had a spacing of 1 Å and dimensions of $77 \times 57 \times 47$ grid units. The boundary conditions were assigned using a 4-Å grid and interpolated over all grid points on the boundary surfaces, using a residue-based potential as described above. The FDPB grid was shifted at every 1000 steps to remain centered on the protein. This change in general destroys the memory of potential map so that a *de novo* FDPB run must be performed. The overhead of the *de novo* run and the update of secondary nonbonded list to the simulation is minimal because the *de novo* run is only about 10 times slower than an update run and the costs are averaged over 1000 steps.

Simulation with the DDD Method

The stochastic dynamics simulation with DDD used an atomic interaction cutoff distance of 15 Å. The nonbonded list was updated every 10 steps, with a secondary nonbonded list of cutoff distance 20 Å. As above, the secondary nonbonded list was updated every 1000 steps to minimize the overhead of the update. The nonbonded interaction routine was optimized to take advantage of the square root-free feature in the DDD solvent model. A reference run was also performed in CHARMM 26 and similar CPU efficiency was observed for both our method and CHARMM

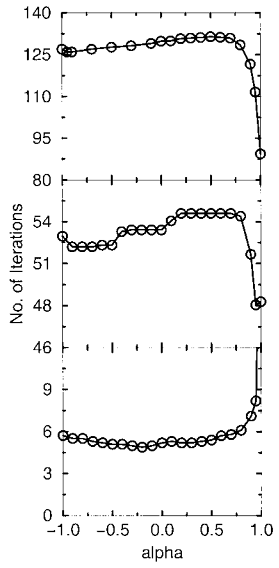

Figure 1. (a) α vs. number of iterations used to reach convergence for the test system, HIV Protease. The finite difference grid spacing is 1.0 Å, and the dimensions are $101 \times 101 \times 101$. Each data point is the average of 10 *de novo* runs. The convergence criterion is 10^{-6}. (b) The convergence criterion is 10^{-3}. The rest is the same as in (a). (c) Each data point is the average of 10 update runs. The rest is the same as in (b).

for a test DDD simulation when the same simulation conditions were given in both methods.

Molecular Model

A polar-hydrogen-only model of HIV Protease (pdb code: 1HPX)[79] was built with Quanta 98.[80] All waters and ligand atoms were removed from the original PDB structures. The total number of atoms is 1844 with 1394 charged atoms after CHARMM parm19 parameters[81] were assigned. The protein structures were briefly energy minimized by 200 steps of steepest descent in Quanta98 with CHARMM to relieve stress.

Results

This section describes the quality of the improvements made to the FDPB core iteration routine, first by implementing the MICCG algorithm, then by introducing approximations; notably relaxing the convergence criterion, utilizing the prior FDPB solution as a first guess for the next calculation in dynamics and energy minimization, updating FDPB forces less frequently than the other molecular mechanics force-field terms, and applying electrostatic focusing. The subsequent section describes the improvements made to the FDPB supporting routines. The final section examines the performance of MD simulations that incorporate all of the enhancements described here.

Acceleration of the FDPB Core Iteration Routine MICCG vs. ICCG

The performance of MICCG vs. ICCG as a function of α is shown in Figure 1 for the HIV Protease test case. Note that the method reverts to ICCG when $\alpha = 0$.

It is evident that MICCG indeed improves the convergence behavior for *de novo* runs (the top and middle panels), but the improvement is not as impressive as demonstrated for model systems[72] where the speedup could be a factor of 2. Also, the optimal value of α is 1.0 instead of 0.95, as was observed for model systems. This value seems to be independent of the convergence criterion used for the *de novo* runs.

For update runs, it turns out that MICCG is not as good as the original ICCG method, as can be seen from the bottom panel of Figure 1. Although the optimal value of α is -0.20, the improvement over standard ICCG is small. It should be pointed out that negative α values were not tested in the literature, and it is not clear why negative values are better than the positive values for some of the cases.

When MICCG and Eisenstat's optimizations are implemented, the FDPB core iteration routine takes 43.8 s on one CPU of a dual-CPU SGI Octane workstation with 384-MB main memory to converge a $101 \times 101 \times 101$ grid to 10^{-6} for HIV Protease, a protein with over 1800 atoms. The CPU used is a 250 MHz IP30 R10000 with 1-MB cache memory. The SGI FORTRAN compiler optimization flag is set to level 2. The core iteration routine inside UHBD takes 170.8 s to reach the same convergence under the same testing conditions, indicating a 3.9 speedup for the present algorithm. We also studied the speedup vs. the size of the system, as shown in Table 2. The results indicate that the gain of the new implementation is greater for larger systems, though the speedup reaches a limit of 3.8 to 3.9.

Adjustment of the Convergence Criteria

To investigate the effects of convergence criterion, we first studied its influence on the electrostatic potential map of protein G (pdb code: 1PGA[82]). Figure 2 shows the electrostatic potential maps of protein G using four different convergence criteria, 10^{-6}, 10^{-3}, 10^{-2}, and 10^{-1}. The comparison indicates that 10^{-2} is sufficiently good, not to say the map using 10^{-3}. However, the map using 10^{-1} is clearly inaccurate.

Table 2. Timings for the Core Iteration Routines in UHBD and the Current MICCG Implementation.

Size of system	UHBD (s)	MICCG (s)	Speedup
$26 \times 26 \times 26$	0.36	0.13	2.8
$51 \times 51 \times 51$	9.6	3.0	3.2
$101 \times 101 \times 101$	170.8	43.8	3.9
$201 \times 201 \times 201$	3168.8	827.3	3.8

All runs are converged to 10^{-6}. CPU times reported are averaged over 10 runs to even out fluctuations; the standard deviations of all means are less than 5% of the means.

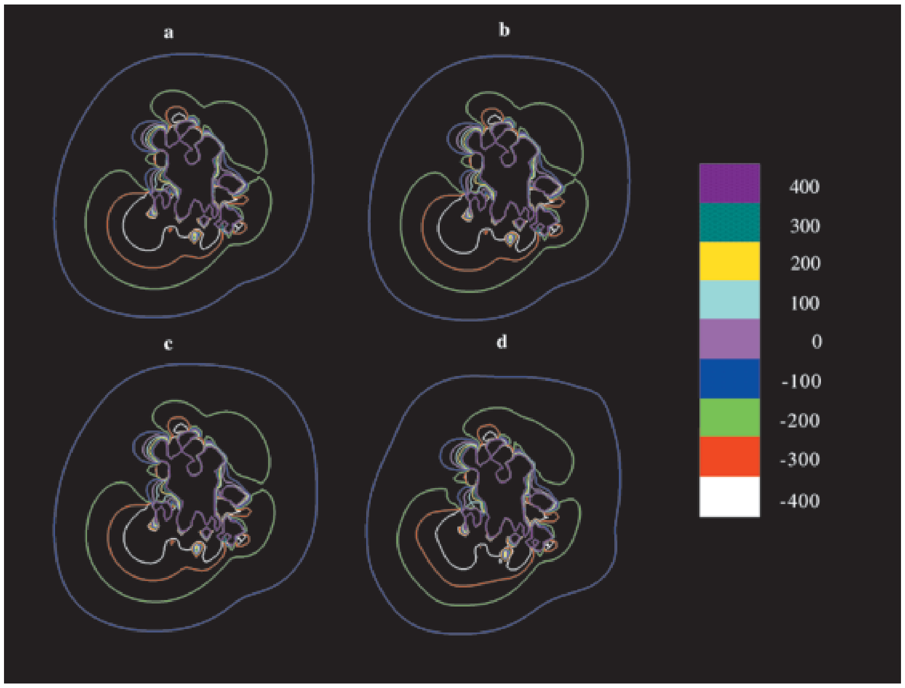

Figure 2. Influence of convergence criterion on the electrostatic potential maps of protein G. a: Convergence criterion 10^{-6}; b: 10^{-3}; c: 10^{-2}; d: 10^{-1}.

The effect of the convergence criterion on electrostatic energies and forces was then investigated. For the test case, HIV Protease, we found that the RMS deviation of the 1394 nonzero electrostatic forces between those using the 10^{-3} convergence criterion and those using 10^{-6} is only 2.5×10^{-6} kcal/mol-Å. The RMS relative deviation is less than 10^{-6}, and the correlation coefficient (c.c.) is over 0.999999. The RMS deviation of 100 total electrostatic energies that we tested is 3.1×10^{-1} kcal/mol, the RMS relative deviation is less than 10^{-6}, and the c.c. is over 0.999999. When using 10^{-2} as convergence criterion, the correlation of forces and energies is still very good as shown before[58] (c.c. >0.99). However, 10^{-1} is rather inaccurate, with c.c. only about 0.9. When a relaxed convergence criterion is applied, the CPU time used by MICCG is dramatically reduced, as shown in Table 3.

Therefore, by utilizing the MICCG algorithm with careful coding and relaxing the convergence criteria to 10^{-2}, we are able to converge the current core iteration at a fraction of the cost of its UHBD counterpart with the default setting 10^{-6}. This is without loss of apparent accuracy in potential, energies, and forces (see Table 2 for the CPU usage of the core iteration inside UHBD for the same test system). In the following, we further address the performance of MICCG in dynamics calculations.

Table 3. Speedup Gained by Using a Less Stringent Convergence Criterion.

Convergence	Iter.	CPU (s)
10^{-6}	128	43.8
10^{-3}	53	18.4
10^{-2}	20	7.3

Convergence: convergence criterion. Iter.: number of iterations needed to achieve the given convergence. CPU: CPU time for the calculation. The test system is a grid of $101 \times 101 \times 101$ for HIV Protease.

Speedup in Molecular Dynamics Simulations

As pointed out in Methods, using the FDPB solution from a recent molecular dynamics (MD) time step as the starting guess for the current time step may be very helpful in speeding FDPB calculations in MD calculations. The same holds in energy minimization calculations. Figure 3 shows that the number of iterations required to achieve convergence slowly approaches the number required for a *de novo* run (53—see Table 3), as the FDPB update run is performed more and more time steps away from the last FDPB solution. The figure also shows that the dynamics simulation does not go far away even after 1000 steps (1 ps) because the memory of the potential map is still quite strong: it still takes fewer than half of the iteration cycles of the *de novo* run to converge the potential map.

However, the speed-up of update runs varies with the convergence criterion used. This is shown in Table 4, which presents the mean number of iterations needed for an update run and a *de novo* run for three different convergence criteria.

Indeed, an update run immediately following the previous potential map can be 10 times as fast as the *de novo* run using the convergence criterion 10^{-3}. However, when using 10^{-6}, the saving is only 32%. At convergence criterion 10^{-3}, the number of

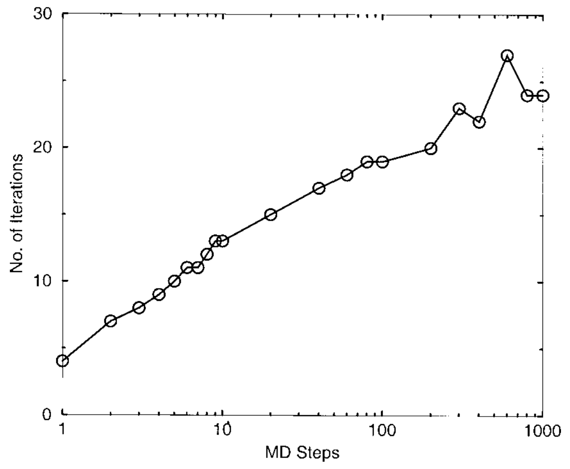

Figure 3. Number of iterations required to reach convergence vs. number of dynamics steps between FDPB calculations for a dynamics trajectory. Convergence criterion is 10^{-3}. All data are averaged over ten experiments using the same grid as mentioned before. The α value is -0.20 for the update runs.

iterations is about five steps for a grid of 101 × 101 × 101, and the CPU time for the core iteration is now about 3 s on the SGI R10000 CPU mentioned above. This is about 13 times longer than a simulation with a distance-dependent dielectric solvent method for the same system with a cutoff of 15 Å (see below).

Intermittent Updating of FDPB Forces in Simulations

As pointed out in Methods, the small change in structure from one MD time step to the next suggests that it may not be necessary to perform FDPB at every step. The accuracy of this approximation was assessed by comparing the forces computed at every step with forces computed every 2–10 steps. Figure 4 shows the results—RMS & c.c.—averaged over 10 tests.

We find that, for a 1-Å grid, updating the FDPB forces every 10 steps yields a relative error of about 20% and a correlation coefficient of 0.92. A similar pattern is observed for calculations with a finer grid of 0.5 Å (data not shown). However, Figure 3 indicates that updating PB forces every two steps (i.e., one step of dynamics

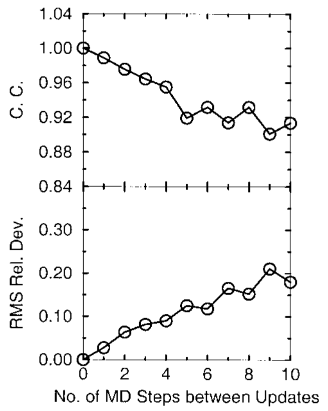

Figure 4. The error in not updating PB forces at every step. c.c: correlation coefficient; RMS Rel. Dev.: RMS relative deviation. The standard deviations of the mean RMS are less than 1%, the standard deviations of the mean c.c. are less than 0.05.

between two FDPB updates) is still quite accurate. Indeed, there is a trade-off: updating FDPB forces less frequently increases the number of iterations needed for each FDPB update (see Fig. 3). Therefore, updating FDPB every step may not be a bad idea if the supporting routines are fast. The efficiency of the supporting routines is discussed below.

Table 4. Number of Iterations Needed for *de novo* and Update Runs as a Function of the Convergence Criterion.

	Mean no. of iterations	
Convergence	*De novo*	Update
10^{-6}	43.8	33.1
10^{-3}	23.9	2.4
10^{-2}	7.3	1.0

All data are obtained by averaging over 10 experiments using the same grid as mentioned before. The α value is 1.00 for the *de novo* runs, -0.20 for the update runs. Note that an update run uses the FDPB solution from the immediate previous time step.

Electrostatic Focusing

For the sample system studied, HIV Protease, if we use a grid of 100 Å^3 with the grid spacing of 2 Å for the first pass, and a final grid of 80 × 60 × 50 Å^3 with grid spacing of 1 Å, the MICCG core routine only takes 0.9 s for a FDPB update run, yet it still uses a high-quality boundary condition. Therefore, electrostatic focusing makes the FDPB core routine faster by another factor of 3–4 (see Table 3). Now, the FDPB core iteration routine takes barely three times as much CPU time as a DDD simulation with the 15-Å cutoff. Note that the calculations with FDPB account for all charge–charge interactions within the protein because the finite-difference potentials include not only the potential due to the solvent, but also long-ranged Coulombic potentials generated by protein atoms. The DDD calculations, in contrast, cut off all electrostatic interactions beyond a range of 15 Å.

Supporting Routines

As described in Methods, the supporting routines are as important as the core iteration routine in determining efficiency. The first bottleneck is the boundary condition assignment. We use two approximations to speed up the process: using residue-based potentials and computing boundary potentials on a coarse grid and then interpolating the boundary potentials to the final grid resolution. These two approximations yield energies and forces that still correlate well with those based on the more rigorous atom-based boundary conditions. Thus, for 10 representative calculations, the RMS relative deviations of energies and atomic forces were on the orders of 10^{-8} and 10^{-6}, respectively, relative to the more rigorous atom-based method. The computation time is reduced from 294 to 0.7 s.

The second bottleneck is the computation of the energies and forces. By an efficient implementation of the Luty and McCammon method and using a cutoff of 8 Å for a grid spacing of 1.0 Å to correct short-range finite difference Coulombic interactions, we are able to reproduce the total electrostatic energy and forces from the implementation in UHBD and achieve high efficiency at the same time. Thus, for 10 representative calculations, the RMS relative deviations of energies and forces were on the orders of 10^{-13} and 10^{-12}, respectively, relative to the UHBD implementation, and the computation time is reduced from 283 to 1.4 s.

Efficiency of the Resulting FDPB Method for Dynamics Simulations

In this section, we demonstrate the efficiency of the current FDPB implementation in the context of dynamics simulations. Two 10,000-step stochastic dynamics (SD) simulations were performed to study the timing of the current implementation with HIV Protease as a test case. The first simulation uses our new FDPB method with 1.0-Å grid spacing. The second simulation uses a distant dielectric constant model ($\varepsilon = 4r$) to treat solvation, and serves as a timing reference. To have a fair comparison with the DDD method, the FDPB forces were updated at every step, although it is acceptable to update forces every other steps as shown above. Updating forces less frequently will further reduce the CPU time of SD/PB simulations.

Timing was conducted on the CPU mentioned above. The SD/PB run used 9,554.59 s, and the SD/DDD run used 2,240.70 s, so the CPU time for SD/PB is only four times as long as that of the SD/DDD simulation. Of the total time spent for SD/PB, about 40.0% is used for the MICCG core iteration routine, 35% is used for the setup of the grid and boundary conditions, and 15% is used for the computation of forces and energy from the potential map. Pairwise nonbonded interactions and pairlist updating take less than 10% of the CPU time.

Discussion and Conclusion

It is interesting to note that a preconditioned conjugate gradient solver gives rather impressive performance for both static and dynamic systems when carefully coded and used. This is achieved by implementing MICCG with Eisenstat's two optimizations, utilizing the electrostatic update in simulations, and applying prudent approximations, including: relaxing the convergence criterion, not updating PB-related forces every step, and using electrostatic focusing. It is also possible to markedly accelerate the supporting routines. The resulting FDPB algorithm delivers efficiency comparable to the distance-dependent-dielectric model for the system tested, making it a strong candidate for solution-phase molecular dynamics simulations. Note, too, that the FDPB method includes all intrasolute electrostatic interactions, whereas the DDD calculations use a 15-Å cutoff. A recent article shows that a simulation of GB solvent model for molecular dynamics is about six times slower than a DDD simulation[56] for a smaller protein, Protein G. It is encouraging that the efficiency of our numerical finite-difference method is comparable to that of the pair-wise GB approximation to the Poisson–Boltzmann method.

There is still room to improve the method of establishing the dielectric boundary. Here we have chosen the van der Waals surface for the sake of simplicity. It appears from recent publications that use of sigmoidal functions to represent atomic volume is an efficient alternative to estimate the dielectric boundary.[61,83] Other possible approaches may be based on the fact that only small conformational change occurs at each step in dynamics simulations. Either way, more efforts are needed to investigate the quality and stability of different choices in the context of FDPB.

Before using any PB continuum solvent with a molecular mechanics force field, a set of atomic cavity radii is needed to reproduce the solvation energies obtained from experiment or computed with explicit solvent models. This is needed to guarantee a balance between solvent–solute and solute–solute interactions. The Roux group have published several works aimed at optimizing these parameters for the CHARMM parameter set.[61,84] Similar approaches are also needed to obtain radii for AMBER and OPLS parameter sets. A related issue is whether FDPB with optimized parameters can reproduce thermodynamic and kinetic quantities for a series of test cases. Much has been done in this regard,[56,57,85–89] although mostly on small organic molecules. With the efficiency of the newly improved FDPB, it is worth investigating more complex systems and longer time scales.

It is of interest to consider how further improvements in efficiency can be achieved when applying FDPB to molecular mechanics simulations. Our dynamics test run shows that further optimization of the core iteration routine, which only takes about 40% of the overall CPU time will no longer provide impressive speedup unless a very different strategy is used to solve the FDPB equation. The performance data also shows that finer grid spacings, such as 0.5 and 0.25 Å are possible with the current algorithm. Once finer grid spacings are used, the core iteration routine will again become the bottle neck. Multigrid approaches may be appropriate for such applications.

Another interesting direction is to implement simulations with the FDPB solvent on multi-CPU platforms. Indeed, a parallel implementation of (M)ICCG has been thoroughly addressed by van der Vorst,[90] who has shown that (M)ICCG performs very well on supercomputers such as Convex and Cray with the indirect indexing technique that he developed.[90]

References

1. Bryngelson, J. D.; Onuchic, J. N.; Socci, N. D.; Wolynes, P. G. Proteins Struct Funct Genet 1995, 21, 167.
2. Karplus, M.; Sali, A. Curr Opin Curr Biol 1995, 5, 58.
3. Dill, K. A.; Chan, H. S. Nat Struct Biol 1995, 5, 58.
4. Shakhnovich, E. I. Curr Opin Struct Biol 1997, 7, 29.
5. Klimov, D. K.; Thirumalai, D. Proc Natl Acad Sci USA 2000, 97, 2544.
6. Honig, B.; Nicholls, A. Science 1995, 268, 1144.
7. Honig, B.; Sharp, K.; Yang, A.-S. J Phys Chem 1993, 97, 1101.
8. Gilson, M. K.; Rashin, A. A.; Fine, R.; Honig, B. J Mol Biol 1985, 183, 503.
9. Gilson, M. K.; Honig, B. Proteins Struct Funct Genet 1988, 3, 32.
10. Gilson, M. K.; Honig, B. Proteins Struct Funct Genet 1988, 4, 7.

11. Warwicker, J.; Watson, H. C. J Mol Biol 1982, 157, 671.
12. Klapper, I.; Hagstrom, R.; Fine, R.; Sharp, K.; Honig, B. Proteins Struct Funct Genet 1986, 1, 47.
13. Gilson, M. K.; Sharp, K. A.; Honig, B. H. J Comp Chem 1988, 9, 327.
14. McQuarrie, D. A. Statistical Mechanics; Harper & Row: New York, 1973.
15. Sharp, K.; Honig, B. Annu Rev Biophys Biophys Chem 1990, 19, 301.
16. Sharp, K. Curr Opin Struct Biol 1994, 4, 234.
17. Gilson, M. K. Curr Opin Struct Biol 1995, 5, 216.
18. Mardis, K.; Luo, R.; David, L.; Potter, M.; Glemza, A.; Payne, G.; Gilson, M. K. J Biomolec Struct Dyn 2000, S1, 89.
19. Kollman, P. A.; Massova, I.; Reyes, C.; Kuhn, B.; Huo, S. H.; Chong, L.; Lee, M.; Lee, T.; Duan, Y.; Wang, W.; Donini, O.; Cieplak, P.; Srinivasan, J.; Case, D. A.; Cheatham, T. E. Acc Chem Res 2000, 33, 889.
20. Davis, M. E.; McCammon, J. A. J Comp Chem 1989, 10, 386.
21. Nicholls, A.; Honig, B. J Comp Chem 1991, 12, 435.
22. Davis, M. E.; Madura, J. D.; Luty, B. A.; McCammon, J. A. Comput Phys Commun 1991, 62, 187.
23. Nicholls, A.; Sharp, K.; Honig, B. Proteins Struct Funct Genet 1991, 11, 281.
24. Tomasi, J.; Persico, M. Chem Rev 1994, 94, 2027.
25. Zauhar, R. J.; Morgan, R. S. J Comp Chem 1990, 11, 603.
26. Zhou, H.-X. J Chem Phys 1994, 100, 3152.
27. Vorobjev, Y. N.; Scheraga, H. A. J Comp Chem 1997, 18, 569.
28. Cortis, C. M.; Friesner, R. A. J Comp Chem 1997, 18, 1591.
29. Holst, M.; Baker, N.; Wang, F. J Comp Chem 2000, 21, 1319.
30. Baker, N.; Holst, M.; Wang, F. J Comp Chem 2000, 21, 1343.
31. Holst, M.; Saied, F. J Comp Chem 1993, 14, 105.
32. Baker, N. A.; Sept, D.; Joseph, S.; Holst, M. J.; McCammon, J. A. Proc Natl Acad Sci USA 2001, 98, 10037.
33. Davis, M. E.; McCammon, J. A. J Comp Chem 1990, 11, 401.
34. Zauhar, R. J. J Comp Chem 1991, 12, 575.
35. Sharp, K. J Comp Chem 1991, 12, 454.
36. Luty, B. A.; Davis, M. E.; McCammon, J. A. J Comp Chem 1992, 13, 768.
37. Niedermeier, C.; Schulten, K. Mol Simulat 1992, 8, 361.
38. Gilson, M. K.; Davis, M. E.; Luty, B. A.; McCammon, J. A. J Phys Chem 1993, 97, 3591.
39. Gilson, M. K.; McCammon, J. A.; Madura, J. D. J Comp Chem 1995, 16, 1081.
40. Wan, S. Z.; Wang, C. X.; Xiang, Z. X.; Shi, Y. Y. J Comp Chem 1997, 18, 1440.
41. Im, W.; Beglov, D.; Roux, B. Comp Phys Commun 1998, 111, 59.
42. Friedrichs, M.; Zhou, R. H.; Edinger, S. R.; Friesner, R. A. J Phys Chem B 1999, 103, 3057.
43. Still, W. C.; Tempczyk, A.; Hawley, R. C.; Hendrickson, T. J Am Chem Soc 1990, 112, 6127.
44. Davis, M. E. J Chem Phys 1994, 100, 5149.
45. David, L.; Field, M. J. Chem Phys Lett 1995, 245, 371.
46. Luo, R.; Moult, J.; Gilson, M. K. J Phys Chem 1997, 101, 11226.
47. Mehler, E. L.; Eichele, G. Biochemistry 1984, 23, 3887.
48. Cramer, C. J.; Truhlar, D. G. Chem Rev 1999, 99, 2161.
49. Bashford, D.; Case, D. A. Annu Rev Phys Chem 2000, 51, 129.
50. Schaefer, M.; Froemmel, C. J Mol Biol 1990, 216, 1045.
51. Hawkins, G. D.; Cramer, C. J.; Truhlar, D. G. Chem Phys Lett 1995, 246, 122.
52. Schaefer, M.; Karplus, M. J Phys Chem 1996, 100, 1578.
53. Qiu, D.; Shenkin, P. S.; Hollinger, F. P.; Still, W. C. J Phys Chem 1997, 101, 3005.
54. Scarsi, M.; Apostolakis, J.; Caflisch, A. J Phys Chem 1997, 101, 8098.
55. Jayaram, B.; Liu, Y.; Beveridge, D. L. J Chem Phys 1998, 109, 1465.
56. Dominy, B. N.; Brooks, C. L. J Phys Chem B 1999, 103, 3765.
57. Tsui, V.; Case, D. A. J Am Chem Soc 2000, 122, 2489.
58. David, L.; Luo, R.; Gilson, M. K. J Comp Chem 2000, 21, 295.
59. Mardis, K. L.; Luo, R.; Gilson, M. K. J Mol Biol 2001, 309, 507.
60. Srinivasan, J.; Trevathan, M. W.; Beroza, P.; Case, D. A. Theor Chem Acc 1999, 101, 426.
61. Nina, M.; Beglov, D.; Roux, B. J Phys Chem B 1997, 101, 5239.
62. Smith, B. J.; Hall, N. E. J Comp Chem 1998, 20, 1482.
63. Sitkoff, D.; Sharp, K. A.; Honig, B. J Phys Chem 1994, 98, 1978.
64. Simonson, T.; Brunger, A. T. J Phys Chem 1994, 98, 4683.
65. Madura, J. D.; Davis, M. E.; Gilson, M. K.; Wade, R. C.; Luty, B. A.; McCammon, J. A. Rev Comput Chem 1994, 5, 229.
66. Edmonds, D. T.; Rogers, N. K.; Sternberg, M. J. E. Mol Phys 1984, 52, 1487.
67. Davis, M. E.; McCammon, J. A. J Comp Chem 1991, 12, 909.
68. Stoer, J.; Bulirsch, R. Introduction to Numerical Analysis; Springer-Verlag: New York, 1993.
69. Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; Vetterling, W. T. Numerical Recipes. The Art of Scientific Computing; Cambridge University Press: Cambridge, 1989.
70. Gustafsson, I. BIT 1978, 18, 142.
71. Meijerink, J. A.; van der Vorst, H. A. J Comp Phys 1981, 44, 134.
72. van der Vorst, H. A. SIAM J Sci Comp 1989, 10, 1174.
73. Eisenstat, S. C. SIAM J Sci Comp 1978, 18, 142.
74. Richards, F. M. Annu Rev Biophys Bioeng 1977, 6, 151.
75. Sanner, M. F.; Olson, A. J.; Spehner, J. C. Biopolymers 1996, 38, 305.
76. Grant, J. A.; Pickup, B. T. J Phys Chem 1995, 99, 3503.
77. van Gunsteren, W. F.; Berendsen, H. J. C. Mol Simulat 1988, 1, 173.
78. Loncharich, R. J.; Brooks, B. R.; Pastor, R. W. Biopolymers 1992, 32, 523.
79. Baldwin, E. T.; Bhat, T. N.; Gulnik, S.; Liu, B. S.; Topol, I. A.; Kiso, Y.; Mimoto, T.; Mitsuya, H.; Erickson, J. W. Structure 1995, 3, 581.
80. Molecular Simulations Inc. San Diego, CA, 1998.
81. Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. J Comp Chem 1983, 4, 187.
82. Gallagher, T.; Alexander, P.; Bryan, P.; Gilliland, G. L. Biochemistry 1994, 33, 4721.
83. OpenEye Scientific Software. Santa Fe, NM, 1999.
84. Nina, M.; Im, W.; Roux, B. Biophys Chem 1999, 78, 89.
85. Marrone, T. J.; Gilson, M. K.; McCammon, J. A. J Phys Chem 1996, 100, 1439.
86. Osapay, K.; Young, W. S.; Bashford, D.; Brooks, C. L., III; Case, D. A. J Phys Chem 1996, 100, 2698.
87. Bashford, D.; Case, D. A.; Choi, C.; Gippert, G. P. J Am Chem Soc 1997, 119, 4964.
88. Scarsi, M.; Apostolakis, J.; Caflisch, A. J Phys Chem B 1998, 102, 3637.
89. Mahadevan, J.; Lee, K. H.; Kuczera, K. J Phys Chem B 2001, 105, 1863.
90. van der Vorst, H. A. Comp Phys Commun 1989, 53, 223.