JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

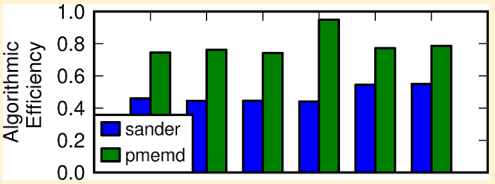
Improving the Efficiency of Free Energy Calculations in the Amber Molecular Dynamics Package

Joseph W. Kaus,*[,†] Levi T. Pierce,[†,‡] Ross C. Walker,[†,‡] and J. Andrew McCammon[†,§,∥,⊥]

[†]Department of Chemistry and Biochemistry, [‡]San Diego Supercomputer Center, [§]Center for Theoretical Biological Physics, [∥]Department of Pharmacology, and [⊥]Howard Hughes Medical Institute, University of California San Diego, La Jolla, California 92093-0365, United States

Supporting Information

ABSTRACT: Alchemical transformations are widely used methods to calculate free energies. Amber has traditionally included support for alchemical transformations as part of the *sander* molecular dynamics (MD) engine. Here, we describe the implementation of a more efficient approach to alchemical transformations in the Amber MD package. Specifically, we have implemented this new approach within the more computationally efficient and scalable *pmemd* MD engine that is included with the Amber MD package. The majority of the gain in efficiency comes from the improved design of the calculation, which includes better parallel scaling and reduction in the calculation of redundant terms. This new implementation is able to reproduce results from equivalent simulations run with the existing functionality but at 2.5 times greater computational efficiency. This new implementation is also able to run softcore simulations at the λ end states making direct calculation of free energies more accurate, compared to the extrapolation required in the existing implementation. The updated alchemical transformation functionality is planned to be included in the next major release of Amber (scheduled for release in Q1 2014), available at http://ambermd.org, under the Amber license.



1. INTRODUCTION

Free energy calculations[1] have been used for many different applications including relative binding energy calculations for drug design,[2,3] solvation free energy calculations on drug-like molecules,[4] and determining the free energy change upon mutation of an amino acid in a protein.[5−7] There are many methods for conducting free energy calculations[8−10] ranging from inexpensive, but more approximate methods, such as docking,[11] to intermediate methods, such as MM/PBSA,[12,13] to expensive, but less approximate methods, such as thermodynamic integration (TI)[14,15] and the multistate Bennett acceptance ratio estimator (MBAR).[16]

These latter methods are more expensive for a variety of reasons, including the need for unphysical intermediate states, long simulation times, often complicated setup and analysis, and the implementation of the method itself. Work by others has described how to optimize the number of intermediate states and reduce the simulation time while preserving the accuracy of TI and MBAR methods.[17−19] However, the efficiency of the algorithm used to implement alchemical transformations has not been examined. This paper focuses on a more efficient implementation of alchemical transformations in the Amber[20] package. Free energies are calculated from these alchemical transformations using TI or MBAR among other methods.

Amber is a widely used software package for running MD simulations, which has been in development for many years.[21−23] The main software packages for MD simulations in Amber are *sander* and *pmemd*. The *pmemd* module of Amber is a more efficient implementation of some of the features available in *sander*, focused mainly on better parallel scaling and GPU acceleration.[22−26] In the current release (v12) of Amber, alchemical transformations are only available in *sander*. We introduce a more efficient implementation of alchemical transformations in *pmemd*, which includes both algorithmic and performance enhancements. This support for alchemical transformations in *pmemd* is planned to be available as part of the next version of Amber (scheduled for release in Q1 2014).

2. THEORY AND METHODS

2.1. Free Energy Calculations. A thermodynamic cycle is often used in free energy calculations in order to make the calculations computationally tractable. These cycles probe the free energy difference between two states using nonphysical transformations to connect the initial and final states.[2] Each segment in the thermodynamic cycle requires a separate simulation. Figure 1 shows an example of a thermodynamic cycle that is used when calculating relative binding free energies. The alchemical transformation of one ligand to another requires less computational time than the direct calculation of the free energy of binding for each ligand. Free energy is a state function, so the relative binding energy will be the same, regardless of which path is used for the calculation.

Received: April 26, 2013

   dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

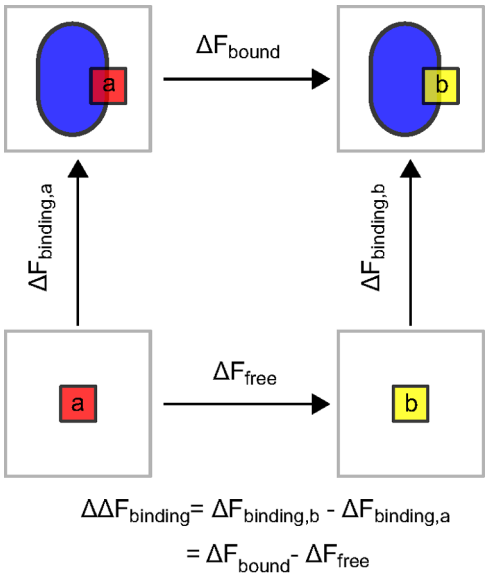


Figure 1. Thermodynamic cycle for a relative binding free energy calculation. The vertical arrows show ligand binding (squares) to a protein (blue oval). The horizontal arrows show the alchemical transformations between the two different ligands bound to the protein (top) and free in solution (bottom). Free energy is a state function, so $\Delta\Delta F_{binding}$ can be calculated using either path.

TI is one method that can be used to calculate the free energy difference for each segment. The free energy is calculated using[14]

$$\Delta F = \int_0^1 d\lambda \left\langle \frac{\partial U}{\partial \lambda} \right\rangle \tag{1}$$

where ΔF is the free energy difference for a segment, U is the potential energy of the system, and λ is a parameter that varies the potential from the initial state where $\lambda = 0$ to the final state where $\lambda = 1$. In practice, eq 1 is integrated numerically, with simulations run at discrete values of λ. In order to determine the error, the correlation time of the $\partial U/\partial\lambda$ values is determined. Then, samples are taken from the complete data set at intervals larger than this correlation time. This subsampling removes correlations in the data, so that the error can be determined using the standard deviation of these values.

Besides TI, free energies can be calculated using MBAR.[16] As with TI, this method makes use of the calculated potential energy at various values of λ. However, rather than integrate eq 1, MBAR uses the additional information of what the potential energy would have been for each configuration calculated at all of the other λ values. To calculate the free energies,[16]

$$\hat{f}_i = -\ln \sum_{j=1}^{K} \sum_{n=1}^{N_j} \frac{\exp[-u_i(x_{jn})]}{\sum_{k=1}^{K} N_k \exp[\hat{f}_k - u_k(x_{jn})]} \tag{2}$$

is solved self-consistently for the free energy $\hat{f}_i$ at each λ value using reduced potential energy u calculated at the all λ values. In practice, eq 2 can be solved using the freely available pyMBAR program. This program can be downloaded from https://simtk.org/home/pymbar. The statistical uncertainty can also be calculated using pyMBAR. This method has been shown to minimize the variance in the calculated free energies, by making more efficient use of the simulation data.[16−28] Both TI and MBAR are post processing methods that can be applied to alchemical transformation simulations. In both the *sander* and *pmemd* implementations, the extra energies needed for MBAR are calculated during the simulation, with little additional cost compared to an equivalent TI simulation.

The potential energy varies smoothly from the initial to the final state, using linear scaling or softcore terms. The general functional form for the potential energy is

$$U(q, \lambda) = U_{common}(q) + (1 - \lambda)U_{i,perturbed}(q, \lambda) + \lambda U_{f,perturbed}(q, \lambda) \tag{3}$$

where U_{common} is the potential for the unperturbed atoms, $U_{i,perturbed}$ and $U_{f,perturbed}$ are the potentials that correspond to the initial and final states for the perturbed part of the system, and q represents the $3N$ atomic coordinates. The potentials for the initial and final states may include softcore van der Waals (vdW) and electrostatic (EEL) terms, which improve the efficiency and stability of the simulations.[29−31]

These potentials only modify the functional form of the intermediate states; at the initial or final states (end states), the potential energy is calculated using the original vdW and EEL potential terms. A free energy calculation may make use of both softcore potentials in a single-step transformation, or may just use the vdW softcore term in a multistep transformation, with the free energy associated with removing the charges calculated in separate simulations. Both methods have been used in this work.

2.2. Implementation. In this section, the differences between the current (*sander*) and new (*pmemd*) implementations of alchemical transformations are discussed. This discussion applies to both TI and MBAR post processing methods, as the same simulation can be analyzed with both methods.

2.2.1. Existing Implementation. In the current (Amber 12) implementation, which uses the *sander* MD engine, a separate topology is created for the initial and final states of the system. Effectively, the code starts two simulations, one for the initial state and another one for the final state. For a simulation that does not use softcore potentials, the potential energy is combined using

$$U(q, \lambda) = (1 - \lambda)[U_{i,common}(q) + U_{i,perturbed}(q, \lambda)] + \lambda[U_{f,common}(q) + U_{f,perturbed}(q, \lambda)] \tag{4}$$

which is equivalent to eq 3. Amber uses a pairwise potential, so the above terms describe the pairwise interactions between atoms.[21] U_{common} describes the pairwise interactions between atoms that are the same in the initial and final states. $U_{i,perturbed}$ and $U_{f,perturbed}$ describe the interactions between the common and perturbed atoms, as well as the interactions among the perturbed atoms. For a simulation that includes softcore potentials, eq 4 is modified so that the softcore atoms become decoupled from the rest of the system, while the bonding terms for the softcore atoms remain intact. The potential used for this case is

dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

$$U(q,\lambda) = (1-\lambda)\left[\frac{1}{1-\lambda}U_{i,\mathrm{bsc}}(q,\lambda) + U_{i,n\mathrm{bsc}}(q,\lambda) + U_{i,\mathrm{perturbed}}(q,\lambda) + U_{i,\mathrm{common}}(q)\right] + \lambda\left[\frac{1}{\lambda}U_{f,\mathrm{bsc}}(q,\lambda) + U_{f,n\mathrm{bsc}}(q,\lambda) + U_{f,\mathrm{perturbed}}(q,\lambda) + U_{f,\mathrm{common}}(q)\right] \quad (5)$$

where U_{bsc} is the potential for the bonded interactions of the softcore atoms and U_{nbsc} is the potential for the nonbonded interactions including the softcore atoms.[31]

Note that in eqs 4 and 5 the U_{common} term is calculated for both the initial and final states. This calculation is redundant, as U_{common} will always be the same for both states. In typical alchemical free energy calculations, the number of perturbed atoms is small compared to the total number of atoms in the system. In this case, the number of pairwise interactions between common atoms is much larger than the number of pairwise interactions involving the perturbed atoms or softcore atoms. In other words, the calculation of U_{common} is expensive compared to the rest of the potential calculations needed for the simulation.

The algorithmic efficiency can be quantified by comparing the speed of the free energy calculation, in terms of the number of nanoseconds that can be simulated in a day, to a regular MD simulation (reference simulation) for either the initial or final state using the same number of cores. In the current *sander* implementation, the calculation is approximately 50% efficient, as there are effectively two simulations running, one for each end state.

The algorithmic efficiency can be improved by only calculating U_{common} once for each step rather than twice. This reduces the number of terms in the direct calculation and also reduces the communication overhead between MPI threads. For each step, the forces that correspond to U_{common} have to be sent across nodes and combined using eqs 4 or 5. This can be detrimental to the scalability of the program. Only the forces for perturbed atoms need to be communicated and combined. These changes are the basis for the new implementation of alchemical transformations in *pmemd*. The idea to remove redundant calculations has been implemented in other MD programs, such as Gromacs.[32] However, this is the first time that this optimization has been applied in the Amber suite of programs. We also introduce an additional optimization, which may be beneficial for vdW-only free energy calculations.

2.2.2. New Implementation. In the new (*pmemd*) implementation, only one topology is used, which contains the common atoms and the atoms corresponding to the end states. This "merged" topology removes the redundant parameters involving the common atoms but still contains all of the parameters for the perturbed atoms. In this work, a dual topology was used; however, the implementation supports both single and dual topology approaches.[33] Creation of this topology is straightforward and is discussed in the Supporting Information. Only a single simulation is run, with all of the terms that differ between the end states stored in separate arrays. At the end of each step, these arrays are combined, giving the same results as the *sander* version. However, the functional form of the potential energy is different, as only one set of calculations is carried out for the common interactions. For simulations without softcore atoms, the functional form is

$$U(q,\lambda) = U_{\mathrm{common}}(q) + (1-\lambda)U_{i,\mathrm{perturbed}}(q,\lambda) + \lambda U_{f,\mathrm{perturbed}}(q,\lambda) \quad (6)$$

which is the same as eq 3. For softcore simulations the potential form is

$$U(q,\lambda) = U_{\mathrm{common}}(q) + U_{i,\mathrm{bsc}}(q,\lambda) + (1-\lambda)[U_{i,\mathrm{nbsc}}(q,\lambda) + U_{i,\mathrm{perturbed}}(q,\lambda)] + U_{f,\mathrm{bsc}}(q,\lambda) + \lambda[U_{f,\mathrm{nbsc}}(q,\lambda) + U_{f,\mathrm{perturbed}}(q,\lambda)] \quad (7)$$

where the difference between eq 5 and eq 7 is that the U_{common} term is only calculated once per time step and there is no division by λ. This reformulation of the potential energy is more efficient as explained above, and allows softcore simulations to be run at the end states where $\lambda = 0$ or $\lambda = 1$. For a softcore simulation run using *sander*, the values of λ cannot be too close to 0 or 1, because the potential energy calculation becomes unstable for low or high λ values when using eq 5. This can make it difficult to accurately determine the free energy difference since simulations cannot be run at the end states and so the values at the end points have to be obtained by extrapolation. One possible work around is to use a method of numerical integration that does not require sampling at the end states which works in most cases but is not always optimal. The new *pmemd* implementation, however, completely removes any limitations on running at the end states, by using eq 7 to calculate the potential energy.

2.3. Simulation Details. All of the simulations follow a similar protocol; any details that are specific to a particular system will be described below. The Amber 12 version of *sander*, patched up to and including bugfix 11, and the modified version of *pmemd* were used for these calculations. Protein models used the Amber 99SB-ILDN force field,[34,35] with the TIP3P[36] model for water. Neutralizing counterions, sodium or chloride, were added to each system as needed. Ligands were parametrized using the generalized Amber force field (GAFF)[37] for the bonded and vdW parameters. Partial charges for ligands were obtained using RESP[38] fitting for the electrostatic potentials calculated using Gaussian03[39] at the Hartree–Fock/6-31G* level of theory. A truncated octahedral periodic box was used with a minimum distance of 15 Å between any box edge and any solute atom. Long range electrostatics were calculated using Particle mesh Ewald (PME),[40] with a 1 Å grid. Short range vdW interactions were truncated at 8 Å with a continuum model long-range correction applied for energy and pressure. Hydrogen bonds were constrained with SHAKE[41] for nonwater molecules and SETTLE[42] for water molecules. This allows for the use of a 2 fs time step.

The same initial coordinates were used for both the *pmemd* and *sander* simulations. Initial geometries were minimized using 20 000 steps of steepest descent minimization at $\lambda = 0.5$. These minimized geometries were then used for simulations at all λ values. The number of λ values used varied between the simulations and is discussed below. Each simulation was heated to 300 K over 500 ps using the Langevin thermostat,[43,44] with a collision frequency set to 2 ps^{-1}. The Berendsen barostat[45] was used to adjust the density over 500 ps at constant pressure, with a target pressure of 1 bar and a 2 ps coupling time. 500 ps of constant volume equilibration was followed by five independent

dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX–XXX

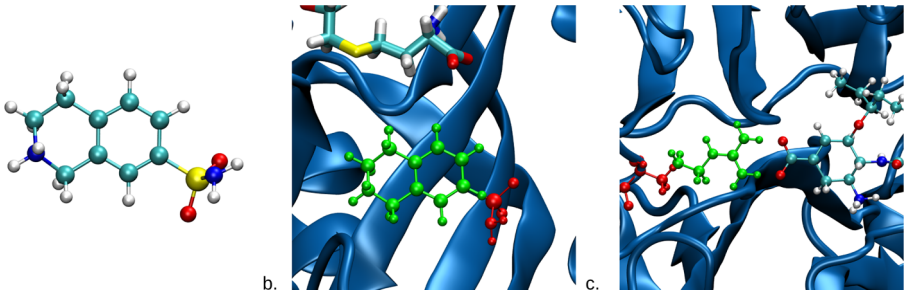


Figure 2. Model systems used in this study are representative of common free energy transformations. All model systems include explicit solvent, which is not shown for clarity. (a) The model for the solvation free energy of the PNMT inhibitor 7-sulfamoyl-1,2,3,4-tetrahydroisoquinolinium. (b) The model for the relative binding energy to PNMT (blue) for the same ligand (red) and the related molecule with the sulfamoyl functional group removed (green). The cofactor is shown in licorice, and the protein is in the new cartoon representation given by STRIDE.[46] (c) The model for the R371A mutation in the active site of NMD: arginine 371 (green); alanine mutant (red). This residue makes a salt bridge with oseltamivir, an inhibitor of NMD. Model images were rendered using the Tachyon module[47] in visual molecular dynamics (VMD).[48]

5 ns long constant volume production simulations. Energies were recorded every 1 ps, and coordinates were saved every 10 ps. Production simulations recalculated the potential energy at each λ value every 1 ps for later analysis with MBAR. Reference simulations were run corresponding to the initial and final states for each system. These were standard MD simulations, so the only difference from the above protocol is that options specific to free energy calculations were disabled. Scaling simulations were run for 100 ps at constant volume, starting from the equilibrated alchemical transformation or reference simulation.

Unless otherwise noted, all simulations were run on 16 cores of the SDSC Gordon Compute Cluster, with dual-socket Intel Xeon E5 8-core 2.6 GHz CPUs in each node and QDR InfiniBand interconnects. The current *sander* implementation requires the core count (MPI tasks) to be a power of two for efficiency while *pmemd* does not have this restriction. Thus, for ease of comparison, we restricted both calculations to use 16 cores.

2.4. Model Systems. In this work, three types of free energy calculations were used to test the new implementation: the solvation free energy, the relative free energy of binding (simply referred to below as the relative binding energy), and the free energy change due to the mutation of a protein residue.

2.4.1. Solvation Free Energy. First, we calculated the solvation free energy for an inhibitor of phenylethanolamine N-methyltransferase (PNMT), which synthesizes epinephrine from norepinephrine.[49] The initial geometry for the inhibitor, 7-sulfamoyl-1,2,3,4-tetrahydroisoquinolinium, was taken from the crystal structure of human PNMT (PDB 1HNN) chain A.[49] The structure is shown in Figure 2a. A single chloride ion was added to neutralize the net charge in the system. The atom types and corresponding partial charges are included in the Supporting Information. The free energy was calculated using

$$\Delta F_{\text{solvation}} = F_{\text{water}} - F_{\text{gas}} \tag{8}$$

where F_{water} is the free energy of the ligand in water and F_{gas} is that of the gas phase ligand. Softcore potentials were used for both vdW and EEL interactions. Nine λ values were used, equally spaced between 0.1 to 0.9. To estimate the error in the resulting free energy caused by omitting the end states, additional simulations were run using the *pmemd* implementation at $\lambda = 0.0$ and $\lambda = 1.0$.

2.4.2. Relative Binding Energy. Next, the relative binding energy of two inhibitors of PNMT was calculated. A thermodynamic cycle was used to calculate the free energy. One segment of the cycle is the difference in free energy with the inhibitors bound, and the other segment is with the inhibitors in solution. This is the same cycle that was shown in Figure 1. This system has been studied previously,[50,51] albeit with a different simulation protocol. Initial coordinates for the bound simulation were obtained from the crystal structure of human PNMT (PDB 1HNN).[49] As in the previous studies,[50,51] the first ligand was 7-sulfamoyl-1,2,3,4-tetrahydroisoquinolinium and the second was 1,2,3,4-tetrahydroisoquinolinium, which is the first ligand with the sulfamoyl moiety removed (Figure 2b). Missing heavy atoms were added using LEaP in AmberTools.[20] The cofactor S-adensoyl-L-homocysteine and ligands were parametrized using the protocol described above. The H++ webserver,[52−54] at http://biophysics.cs.vt.edu/H++, was used to determine the protonation states of the Histidine residues.

The free energy was calculated using

$$\Delta F_{\text{bound}} = F_{\text{ligand2,bound}} - F_{\text{ligand1,bound}}$$
$$\Delta F_{\text{free}} = F_{\text{ligand2,free}} - F_{\text{ligand1,free}} \tag{9}$$

where "bound" refers to the ligands bound to PNMT and "free" refers to the ligands in water. The bound system had five sodium ions added, and the free system had one chloride ion added to neutralize the net charge. As with the solvation free energy calculation, softcore vdW and EEL potentials were used with the same λ values.

2.4.3. Mutation of a Protein. Finally, the free energy difference between a wild-type and mutant protein was calculated. The protein studied is N1 neuraminidase (NMD), an influenza surface protein, bound to oseltamivir, an anti-influenza drug. NMD has been the subject of structural and theoretical studies.[55−57] Arginine 371, a residue in the active site that interacts directly with the drug oseltamivir, is mutated to alanine (Figure 2c). This could be used as part of a thermodynamic cycle to determine how this mutation affects drug binding. However, we were only interested in determining the accuracy of our new implementation, and so, the additional simulations required for a complete thermodynamic cycle were not performed.

The initial coordinates for this system came from the crystal structure (PDB 2HU4) chain A.[55] All of the histidine residues were protonated in the ε position. Disulfide bonds were added between cysteine residues based on the information in the crystal structure. Four neutralizing sodium ions were added to the system.

Unlike the previous simulations, the softcore EEL potential was not used, so the calculation was broken down into three steps. In the first step, the charges on residue 371 were removed. Then, the softcore vdW potential was used to mutate arginine to alanine, with no charge on the perturbed atoms. Finally, the charges were restored on the mutated residue. The free energy was calculated using

$$\Delta F_{\text{charge,wt}} = F_{\text{wt,no charge}} - F_{\text{wt,charge}}$$
$$\Delta F_{\text{vdW}} = F_{\text{mut,vdW}} - F_{\text{wt,vdW}}$$
$$\Delta F_{\text{charge,mut}} = F_{\text{mut,no charge}} - F_{\text{mut,charge}} \quad (10)$$

where wt denotes the wild-type NMD and mut denotes the R371A mutant. This multistep method should give the same results as the single-step method, but there may be cases where it converges more rapidly.[30] For the charge removal calculations, 11 equally spaced λ values were used, from λ = 0 to λ = 1. For the vdW calculation, the softcore vdW potential was used, with nine equally spaced λ values from λ = 0.1 to λ = 0.9.

2.5. Accuracy and Efficiency Metrics. In order to determine the accuracy and efficiency of our new implementation, we compared the free energies calculated from *pmemd* simulations to equivalent *sander* simulations. The deviation between the resulting free energies is

$$|\Delta\Delta F| = |\langle \Delta F_{pmemd} \rangle| - |\langle \Delta F_{sander} \rangle| \quad (11)$$

where ΔF_{pmemd} and ΔF_{sander} are the free energies calculated from the alchemical transformations simulated using *pmemd* and *sander*, respectively. The brackets denote an average over independent simulations. The uncertainty in the deviation is

$$\sigma_{\Delta\Delta F} = \sqrt{\sigma^2_{\Delta F_{pmemd}} + \sigma^2_{\Delta F_{sander}}} \quad (12)$$

where $\sigma_{\Delta F}$ is the standard deviation of the free energy over independent simulations. We also define ΔF_{window} as the free energy difference for the alchemical transformation between adjacent λ values. Then eq 11 is used to calculate $|\Delta\Delta F_{\text{window}}|$, the deviation between each pair of λ values.

It is well established that a standard MD simulation will run more quickly in *pmemd* than in *sander* for the same number of cores.[22,23] The metric used to determine the algorithmic efficiency of the new implementation, as opposed to the raw performance, has to be independent of the absolute speed of the calculation in either program. In this work, the algorithmic efficiency is defined as

$$\text{algorithmic efficiency} = \frac{\text{speed of free energy calc.}}{\text{speed of reference calc.}} \quad (13)$$

where the speed is in ns/day and reference calculations refers to classical MD simulations of the end states for the system being studied. The speed was averaged over all λ values for the free energy calculations and over both end states for the reference simulations. This is the key metric, as it shows which implementation is more efficient, regardless of the absolute speed of the program that it is implemented in. The overall change in performance from both improvements in the algorithmic efficiency and the optimization of the code in *pmemd* can be quantified with the relative speed (rel. speed)

$$\text{rel. speed} = \frac{\text{speed of free energy calc. in } pmemd}{\text{speed of free energy calc. in } sander} \quad (14)$$

3. RESULTS AND DISCUSSION

3.1. Comparison of the Calculated Free Energies. The most important assessment of the implementation is to make sure it is correct. Here, we compare the *pmemd* implementation to the widely used *sander* implementation since both programs are part of Amber[20] and provide equivalent results for standard MD simulations. This makes direct comparison of the results straightforward, as it avoids issues arising from differences in the results due to different algorithms or force field parameters, and thus, in principal, it is only limited by convergence of the sampling. As discussed, most of the free energies were calculated without including the end states due to the limitations in the original *sander* implementation. For comparison purposes, we are thus calculating the free energy difference between two nonphysical states λ = 0.1 to λ = 0.9. To estimate the error in the resulting free energy caused by not including the end states, we used the *pmemd* implementation to run alchemical transformations at the end states for the solvation free energy system. When these end states are included, the free energy decreased by 20 kcal/mol, suggesting that these states are important for correctly calculating the free energy. However, our goal with this work is to compare the *sander* and *pmemd* implementations at the λ values that can be used with both implementations.

The deviation in the free energies, calculated using eq 11, is shown in Figure 3. All of the deviations are within 0.5 kcal/mol,

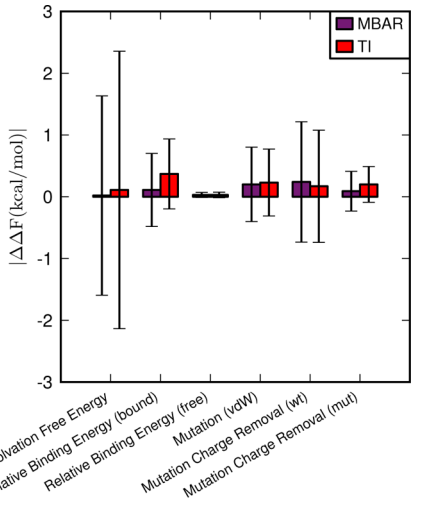


Figure 3. Deviation for each model system, defined as the magnitude of the difference between the free energies calculated using *pmemd* and *sander*, averaged over five independent simulations (eq 11). Error bars represent the propagation of the standard deviation of the free energies calculated from independent simulations (eq 12). The production stage for each window was simulated for 5 ns.

with the largest uncertainty calculated for the solvation free energy system. This system is difficult to converge, as it involves complete decoupling of a ligand from the solvent. These results indicate that the implementation of alchemical transformations in *pmemd* matches the results from the *sander* implementation.

To further examine the deviations in the resulting free energies, $|\Delta\Delta F_{\text{window}}|$, the deviation for each pair of adjacent λ values was calculated for each model system as shown in Figure 4. A per window deviation of 0 kcal/mol indicates that the free energy difference between adjacent λ values is the same for the

dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

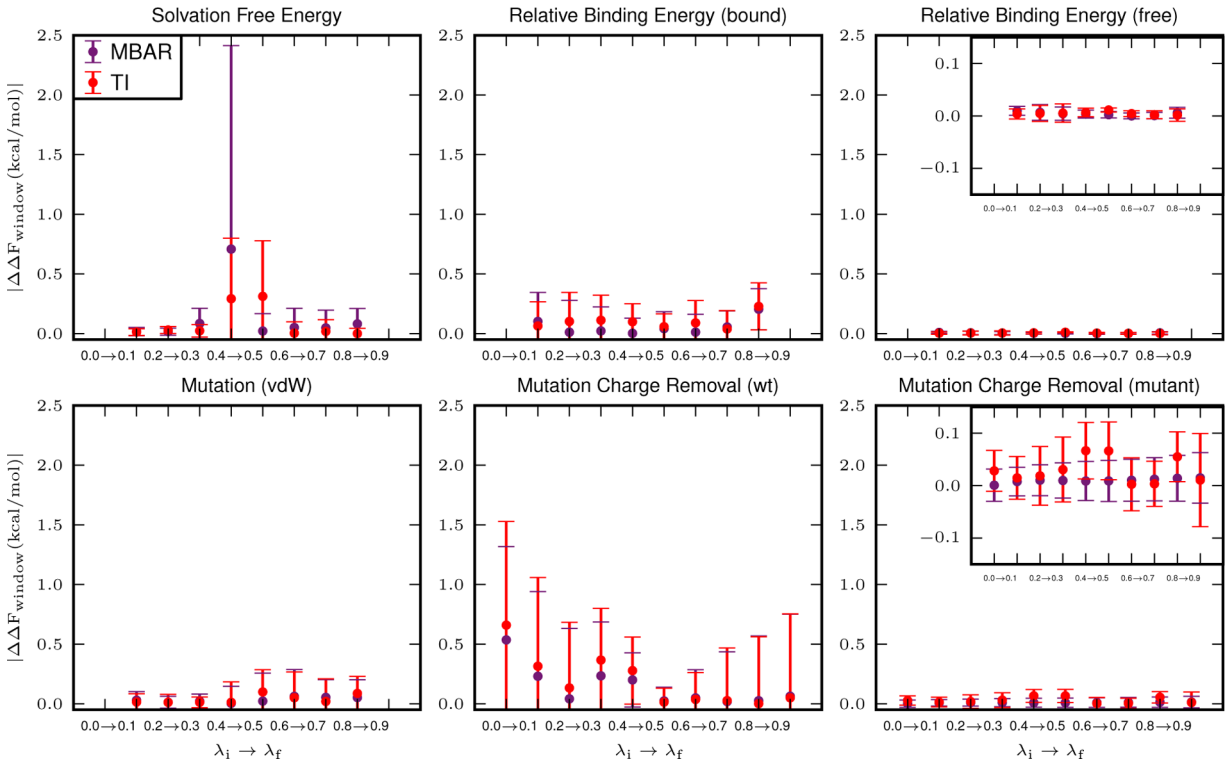


Figure 4. Deviation in the free energy, separated into contributions from each pair of adjacent λ values (windows). The free energy for each window describes the free energy change for the nonphysical transformation from λ_i to λ_f. Each point represents the deviation in this value calculated from *pmemd* and *sander*, averaged over five independent calculations. Error bars represent the propagation of the standard deviation of the free energies calculated from independent simulations (eq 12). The production stage for each window was simulated for 5 ns.

pmemd and *sander* calculations. All of the differences are under 1 kcal/mol, with the largest deviations found with the solvation free energy and wild type charge removal calculations. These results indicate that the deviations are simply due to small differences in which states were sampled in the *pmemd* or *sander* calculations. This issue is inherent to all stochastic simulations, as a simulation with different random velocities will explore different configurations resulting in slightly different free energies.

These model systems are representative of common transformations in free energy calculations. We have shown that the new implementation of alchemical transformations in *pmemd* reproduces the results from equivalent simulations run with *sander*. The small deviation in the resulting free energies is due to the stochastic nature of the simulations, where the perturbed atoms explore slightly different regions of phase space in the *pmemd* and *sander* simulations.

3.2. Algorithmic Efficiency of the New Implementation. The new implementation of alchemical transformations in *pmemd* has been shown to produce results equivalent to the original *sander* implementation. Next, we examined the algorithmic efficiency of the new implementation (Figure 5a). These results show that the original *sander* implementation is approximately 50% efficient, as expected, while the new *pmemd* implementation is between 75% and 95% efficient. The range is due to an additional optimization, which will be discussed below. This corresponds to a gain in algorithmic efficiency of 1.5 to 2 times over the original implementation.

Much of the algorithmic efficiency gain is due to the removal of the redundant calculation of U_{common} in eqs 6 and 7. However, these gains are still limited due to the need to calculate the reciprocal sum for PME[40] twice, once for each end state. This limitation arises from the nonlinearity of the reciprocal sum, so it cannot be decomposed into pairs. In the case of the vdW-only transformation, the mutated residue has had all charges removed, so the value for the reciprocal sum was the same for both end states. In this case, *pmemd* only calculates the reciprocal sum once, leading to a 20% increase in the algorithmic efficiency of the calculation. This applies to all vdW-only transformations and will be automatically detected.

Breaking up a free energy calculation into multiple steps may prevent certain convergence issues that can be seen with single-step transformations. However, a multistep transformation requires three steps, while a single-step transformation requires only one step. The type of transformation to be used will depend on the details of the system under study.[30] It may be beneficial to use a multistep transformation in certain cases, and in these cases, the vdW-only transformation will run more efficiently in *pmemd* than the other transformations.

The relative speed (Figure 5b) shows that using *pmemd* over *sander* reduces the computational cost by a factor of at least 2.5 times regardless of the type of free energy calculation. The additional gain in relative speed seen with the vdW-only transformation is due to the optimization discussed. The algorithmic efficiency gain depends on the number of atoms that are perturbed between the end states. However, for typical alchemical transformations, only a small number of atoms are perturbed to prevent sampling issues. For these systems, there is a clear benefit in using the new, more efficient, *pmemd* implementation.

3.3. Scaling of the New Implementation. The new implementation in *pmemd* is more algorithmically efficient, but it is also important to quantify how well it scales to a large number of cores. We addressed this question by looking at the

dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

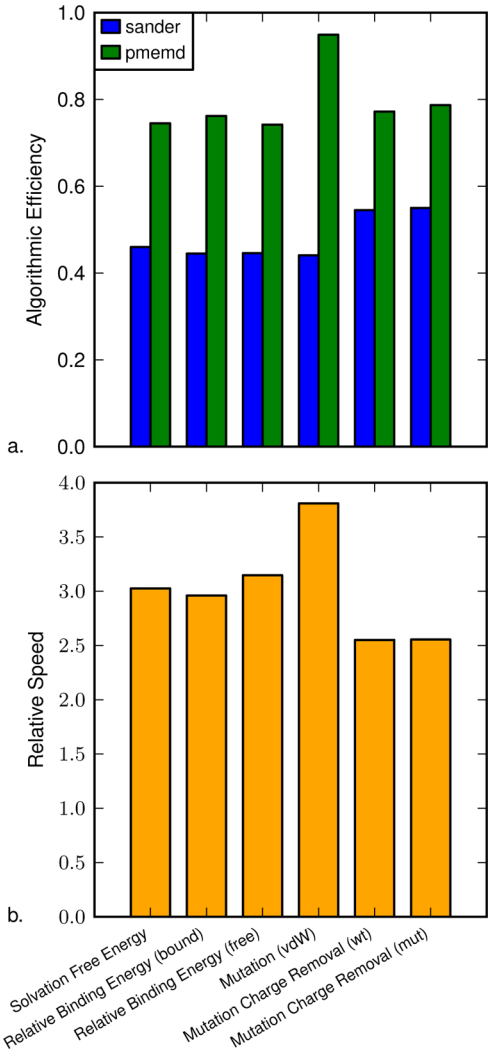


Figure 5. Algorithmic efficiency and relative speed for the model systems. (a) Algorithmic efficiency of the calculation for the model systems using the original *sander* and new *pmemd* implementations. (b) Relative speed, which describes how much faster the free energy calculation was using the *pmemd* implementation.

algorithmic efficiency and relative speed for a single λ window run using different numbers of cores. Specifically, we examined the λ = 0.5 window from the bound state relative binding energy calculation.

As shown in Figure 6, the algorithmic efficiency is stable over a large number of core counts, indicating that it is independent of the number of cores. For very large core counts, the speed of the reciprocal sum becomes a limiting factor, causing a small decrease in the algorithmic efficiency for *pmemd* compared to *sander*. In the original implementation in *sander*, the reciprocal sum is calculated for each end state in parallel, using half of the total number of cores. In the new implementation in *pmemd*, the reciprocal sum is calculated for the end states sequentially, using the total number of cores. This is not as efficient for very large core counts but is a limitation that arises from the current structure of the parallel code in *pmemd*. However, the relative speed jumps from three to five times and the absolute speed for *pmemd* increases more rapidly than *sander*, indicating that an alchemical transformation simulated using *pmemd* will use fewer computational resources over a large range of core counts.

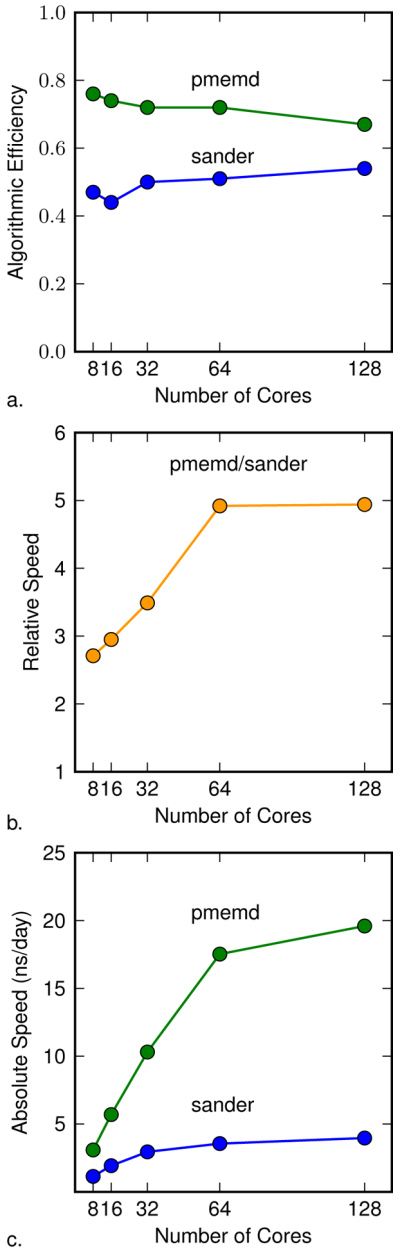


Figure 6. Scaling for the λ = 0.5 point taken from the bound state of the relative binding energy model system. (a) Algorithmic efficiency for the original *sander* and new *pmemd* implementations as the number of cores is varied. (b) Relative speed using *pmemd* over *sander* for the same calculation at different core counts. (c) Absolute speed of the same calculation in nanoseconds per day at different core counts. Each simulation was run for 100 ps.

These results were determined using the bound state relative binding energy calculation. Similar results would be expected with all of the model systems that include a protein. For the smaller model systems where the ligands were in water, the scaling is not expected to be as efficient due to the small total number of atoms. However, even at low core counts, these systems already run much faster than the larger systems, so additional scaling would not be particularly beneficial for calculating free energies.

dx.doi.org/10.1021/ct400340s | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

4. CONCLUSIONS

An efficient method for alchemical transformation calculations has been implemented into the *pmemd* module of Amber.[20] This implementation produces results that are equivalent to those from the original *sander* program. We show the majority of the improvement in the calculation speed is due to improvements in algorithmic efficiency from the new implementation in *pmemd*. The new implementation can run softcore calculations at $\lambda = 0$ and $\lambda = 1$, facilitating direct calculation of free energies without needing to extrapolate the energy at the end states. The interface is very similar to the *sander* program, and only a few changes are needed in the Amber input files.

The improvements in the algorithmic efficiency and relative speed for alchemical free energy calculations will appeal to many users of Amber. These improvements make more efficient use of computer time and provide a guide for further improvements in the implementation of free energy calculations. Future work will focus on enabling the use of Graphics Processing Units (GPUs)[24−26] to accelerate alchemical free energy calculations, based on this work in *pmemd*. Our new implementation is planned to be released publicly as part of the next version of the Amber software.

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

Assignment of parameter types for ligands, procedure used to generate the input files for the model systems, and calculated free energies for the model systems. This material is available free of charge via the Internet at http://pubs.acs.org/.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: jkaus@ucsd.edu.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors thank Tom Steinbrecher and Mehrnoosh Arrar for useful discussions. Joseph Kaus is supported in part by the Molecular Biophysical Training Grant from the National Institutes of Health (NIH). Additional support is provided by the NIH, National Science Foundation (NSF), Howard Hughes Medical Institute (HHMI), Center for Theoretical Biological Physics (CTBP), and National Biomedical Computation Resource (NBCR). Ross Walker and Levi Pierce acknowledge funding from the National Science Foundation through the Scientific Software Innovations Institutes program NSF SI2-SSE (NSF1047875 and NSF1148276) grants to Ross Walker. Ross Walker additionally acknowledges funding through the NSF XSEDE program and through a fellowship from NVIDIA Inc. Computer time was provided by the San Diego Supercomputer Center under XSEDE award TG-CHE130010 to Ross Walker.

■ REFERENCES

(1) Tembe, B.; McCammon, J. A. *Comp. Chem.* **1984**, *8*, 281−283.
(2) Steinbrecher, T.; Labahn, A. *Curr. Med. Chem.* **2010**, *17*, 767−785.
(3) Jorgensen, W. L. *Science* **2004**, *303*, 1813−1818.
(4) Shivakumar, D.; Williams, J.; Wu, Y.; Damm, W.; Shelley, J.; Sherman, W. *J. Chem. Theory Comput.* **2010**, *6*, 1509−1519.
(5) Wong, C. F.; McCammon, J. A. *Isr. J. Chem.* **1986**, *27*, 211−215.
(6) Kollman, P. *Chem. Rev.* **1993**, *93*, 2395−2417.
(7) Espinoza-Fonseca, L. M. *Biochemistry* **2009**, *48*, 11332−11334.
(8) Ytreberg, F. M.; Swendsen, R. H.; Zuckerman, D. M. *J. Chem. Phys.* **2006**, *125*, 184114.
(9) Mobley, D. L.; Dill, K. A. *Structure* **2009**, *17*, 489−498.
(10) Wereszczynski, J.; McCammon, J. A. *Q. Rev. Biophys.* **2012**, *45*, 1−25.
(11) Kuntz, I. D.; Blaney, J. M.; Oatley, S. J.; Langridge, R.; Ferrin, T. E. *J. Mol. Biol.* **1982**, *161*, 269−288.
(12) Srinivasan, J.; Cheatham, T. E., III; Cieplak, P.; Kollman, P. A.; Case, D. A. *J. Am. Chem. Soc.* **1998**, *120*, 9401−9409.
(13) Kollman, P. A.; Massova, I.; Reyes, C.; Kuhn, B.; Huo, S.; Chong, L.; Lee, M.; Lee, T.; Duan, Y.; Wang, W.; Donini, O.; Cieplak, P.; Srinivasan, J.; Case, D. A.; Cheatham, T. E., III *Acc. Chem. Res.* **2000**, *33*, 889−897.
(14) Kirkwood, J. G. *J. Chem. Phys.* **1935**, *3*, 300−313.
(15) Genheden, S.; Nilsson, I.; Ryde, U. *J. Chem. Inf. Model.* **2011**, *51*, 947−958.
(16) Shirts, M. R.; Chodera, J. D. *J. Chem. Phys.* **2008**, *129*, 124105.
(17) Shirts, M. R.; Pande, V. S. *J. Chem. Phys.* **2005**, *122*, 144107.
(18) Bruckner, S.; Boresch, S. *J. Comput. Chem.* **2011**, *32*, 1303−1319.
(19) Bruckner, S.; Boresch, S. *J. Comput. Chem.* **2011**, *32*, 1320−1333.
(20) Case, D. A.; Darden, T. A.; Cheatham III, T. E.; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Walker, R. C.; Zhang, W.; Merz, K. M.; Roberts, B.; Hayik, S.; Roitberg, A.; Seabra, G.; Swails, J.; Götz, A. W.; Kolossváry, I.; Wong, K. F.; Paesani, F.; Vanicek, J.; Wolf, R. M.; Liu, J.; Wu, X.; Brozell, S. R.; Steinbrecher, T.; Gohlke, H.; Cai, Q.; Ye, X.; Wang, J.; Hsieh, M. J.; Cui, G.; Roe, D. R.; Mathews, D. H.; Seetin, M. G.; Salomon-Ferrer, R.; Sagui, C.; Babin, V.; Luchko, T.; Gusarov, S.; Kovalenko, A.; Kollman, P. A. *Amber 12*; University of California, San Francisco: San Francisco, CA, 2012.
(21) Pearlman, D. A.; Case, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham, T. E., III; DeBolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. *Comput. Phys. Commun.* **1995**, *91*, 1−41.
(22) Case, D. A.; Cheatham, T. E., III; Darden, T.; Gohlke, H.; Luo, R.; Merz, K. M.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J. *J. Comput. Chem.* **2005**, *26*, 1668−1688.
(23) Salomon-Ferrer, R.; Case, D. A.; Walker, R. C. *WIREs Comput. Mol. Sci.* **2012**, *3*, 198−210.
(24) Götz, A. W.; Williamson, M. J.; Xu, D.; Poole, D.; Grand, S. L.; Walker, R. C. *J. Chem. Theory Comput.* **2012**, *8*, 1542−1555.
(25) Salomon-Ferrer, R.; Götz, A. W.; Poole, D.; Grand, S. L.; Walker, R. C. *J. Chem. Theory Comput.* **2013**, accepted.
(26) Grand, S. L.; Götz, A. W.; Walker, R. C. *Comput. Phys. Commun.* **2013**, *184*, 374−380.
(27) Paliwal, H.; Shirts, M. R. *J. Chem. Theory Comput.* **2011**, *7*, 4115−4134.
(28) Tan, Z.; Gallicchio, E.; Lapelosa, M.; Levy, R. M. *J. Chem. Phys.* **2012**, *136*, 144102.
(29) Shirts, M. R.; Pande, V. S. *J. Chem. Phys.* **2005**, *122*, 134508.
(30) Steinbrecher, T.; Joung, I.; Case, D. A. *J. Comput. Chem.* **2011**, *32*, 3253−3263.
(31) Steinbrecher, T.; Mobley, D. L.; Case, D. A. *J. Chem. Phys.* **2007**, *127*, 214108.
(32) Hess, B.; Kutzner, C.; van der Spoel, D.; Lindahl, E. *J. Chem. Theory Comput.* **2008**, *4*, 435−447.
(33) Pearlman, D. *J. Phys. Chem.* **1994**, *98*, 1487−1493.
(34) Hornak, V.; Abel, R.; Okur, A.; Strockbine, B.; Roitberg, A.; Simmerling, C. *Proteins* **2006**, *65*, 712−725.
(35) Lindorff-Larsen, K.; Piana, S.; Palmo, K.; Maragakis, P.; Klepeis, J. L.; Dror, R. O.; Shaw, D. E. *Proteins* **2010**, *78*, 1950−1958.
(36) Jorgensen, W. L.; Chandrasekhar, J.; Madura1, J. D.; Impey, R. W.; Klein, M. L. *J. Chem. Phys.* **1983**, *79*, 926.
(37) Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. *J. Comput. Chem.* **2004**, *25*, 1157−1174.

(38) Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Merz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. *J. Am. Chem. Soc.* **1995**, *117*, 5179−5197.
(39) Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Montgomery,; Jr.,; A., J.; Vreven, T.; Kudin, K. N.; Burant, J. C.; Millam, J. M.; Iyengar, S. S.; Tomasi, J.; Barone, V.; Mennucci, B.; Cossi, M.; Scalmani, G.; Rega, N.; Petersson, G. A.; Nakatsuji, H.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Klene, M.; Li, X.; Knox, J. E.; Hratchian, H. P.; Cross, J. B.; Bakken, V.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Ayala, P. Y.; Morokuma, K.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Zakrzewski, V. G.; Dapprich, S.; Daniels, A. D.; Strain, M. C.; Farkas, O.; Malick, D. K.; Rabuck, A. D.; Raghavachari, K.; Foresman, J. B.; Ortiz, J. V.; Cui, Q.; Baboul, A. G.; Clifford, S.; Cioslowski, J.; Stefanov, B. B.; Liu, G.; Liashenko, A.; Piskorz, P.; Komaromi, I.; Martin, R. L.; Fox, D. J.; Keith, T.; Al-Laham, M. A.; Peng, C. Y.; Nanayakkara, A.; Challacombe, M.; Gill, P. M. W.; Johnson, B.; Chen, W.; Wong, M. W.; Gonzalez, C.; Pople, J. A. *Gaussian 03*; Gaussian, Inc.: Wallingford, CT, 2004.
(40) Darden, T.; York, D.; Pedersen, L. *J. Chem. Phys.* **1993**, *98*, 10089−10092.
(41) Ryckaert, J.-P.; Ciccotti, G.; Berendsen, H. J. C. *J. Comput. Phys.* **1977**, *23*, 327−341.
(42) Miyamoto, S.; Kollman, P. A. *J. Comput. Chem.* **1992**, *13*, 952−962.
(43) Pastora, R. W.; Brooks, B. R.; Szaboc, A. *Mol. Phys.* **1988**, *65*, 1409−1419.
(44) Loncharich, R. J.; Brooks, B. R.; Pastor, R. W. *Biopolymers* **1992**, *32*, 523−535.
(45) Berendsen, H. J. C.; Postma, J. P. M.; van Gunsteren, W. F.; DiNola, A.; Haak, J. R. *J. Chem. Phys.* **1984**, *81*, 3684−3690.
(46) Frishman, D.; Argos, P. *Proteins: Struct., Funct., Genet.* **1995**, *23*, 566−579.
(47) Stone, J. An Efficient Library for Parallel Ray Tracing and Animation. M.Sc. thesis, Computer Science Department, University of Missouri-Rolla, Rolla, MO, 1998.
(48) Humphrey, W.; Dalke, A.; Schulten, K. *J. Mol. Graphics* **1996**, *14*, 33−38.
(49) Martin, J. L.; Begun, J.; McLeish, M. J.; Caine, J. M.; Grunewald, G. L. *Structure* **2001**, *9*, 977−985.
(50) Nair, P. C.; Malde, A. K.; Mark, A. E. *J. Chem. Theory Comput.* **2011**, *7*, 1458−1468.
(51) Riniker, S.; Christ, C. D.; Hansen, N.; Mark, A. E.; Nair, P. C.; van Gunsteren, W. F. *J. Chem. Phys.* **2011**, *135*, 024105.
(52) Gordon, J. C.; Myers, J. B.; Folta, T.; Shoja, V.; Heath, L. S.; Onufriev, A. *Nucleic Acids Res.* **2005**, *33*, W368−W371.
(53) Myers, J.; Grothaus, G.; Narayanan, S.; Onufriev, A. *Proteins* **2006**, *63*, 928−938.
(54) Anandakrishnan, R.; Aguilar, B.; Onufriev, A. V. *Nucleic Acids Res.* **2012**, *40*, W537−W541.
(55) Russell, R. J.; Haire1, L. F.; Stevens, D. J.; Collins, P. J.; Lin, Y. P.; Blackburn, G. M.; Hay, A. J.; Gamblin, S. J.; Skehel1, J. J. *Nature* **2006**, *443*, 45−49.
(56) Lawrenz, M.; Baron, R.; McCammon, J. A. *J. Chem. Theory Comput.* **2009**, *5*, 1106−1116.
(57) Amaro, R. E.; Swift, R. V.; Votapka, L.; Li, W. W.; Walker, R. C.; Bush, R. M. *Nat. Commun.* **2011**, *2*, 388.