MDSimAid: Automatic Parameter Optimization in Fast Electrostatic Algorithms

Michael S. Crocker[1] Scott S. Hampton[1] Thierry Matthey[2]

Jesús A. Izaguirre[1*]

[1]Department of Computer Science and Engineering, University of Notre Dame

Notre Dame, IN 46556, USA

[2]Parallab and Bergen Center for Computational Science, University of Bergen

Bergen, Norway

March 14, 2005

Abstract

MDSimAid is a recommender system that optimizes parallel Particle Mesh Ewald (PME) and both sequential and parallel multigrid (MG) summation fast electrostatic solvers. MDSimAid optimizes the running time or parallel scalability of these methods within a given error tolerance. MDSimAid performs a run-time constrained search on the parameter space of each

[*]Corresponding author: izaguirr@cse.nd.edu

method starting from semi-empirical performance models. Recommended parameters are presented to the user. MDSimAid's optimization of MG leads to configurations that are up to 14 times faster or 17 times more accurate than published recommendations. Optimization of PME can improve its parallel scalability, making it run twice as fast in parallel in our tests. MDSimAid and its Python source code are accessible through a web portal located at http://mdsimaid.cse.nd.edu.

1 Introduction

At each timestep of a molecular dynamics (MD) simulation under periodic boundary conditions (PBC), it is often necessary to solve the following summation to compute the electrostatic potential energy:

$$U^{\text{electrostatic}} = \sum_{i=1}^{N} \sum_{j \notin \chi(i)} \sideset{}{'}\sum_{\mathbf{m} \in \mathbb{Z}^3} \frac{q_i q_j}{4\pi\varepsilon_0 \left|\mathbf{r}_j - \mathbf{r}_i + \mathbf{m}L\right|}, \tag{1}$$

and to compute the electrostatic force,

$$F^{\text{electrostatic}} = -\nabla U^{\text{electrostatic}}, \tag{2}$$

where the sum is over all periodic cells with the integral triplet index $\mathbf{m}$, and self-interactions excluded; ε_0 is the dielectric coefficient; $\mathbf{r}_i$ and q_i are the position and charge of particle i; the set $\chi(i)$ contains atoms to be excluded from calculation with atom i, including those that are covalently bonded to i; and L is the length of a periodic cell of volume L^3. Eq. (1) gives a conditionally convergent, infinite summation over all periodically replicated cells. Conditional convergence means that the sum may or may not converge depending on the order in which the summation is performed. A physically meaningful interpretation is given in [1].

Direct computation of all interacting pairs of N particles leads to an $O(N^2)$ algorithm. Significant effort has been put into reducing the computational cost of calculating these interactions. The Ewald [2] and particle mesh Ewald (PME) [3, 4] methods have been optimized to run in $O(N^{3/2})$ and $O(N \log N)$ time, respectively. The fast multipole (FMM) [5] and the multigrid summation (MG) [6–8] methods have improved the asymptotic running time to $O(N)$.

While these fast solvers are asymptotically quicker, in practice there are numerous parameters to configure for optimal performance depending on the system size and desired accuracy. An optimal choice of parameters cannot be fully accomplished analytically due to the complexity of the methods. However, there are many studies that propose schema for determining parameter values for various fast force evaluators [9–17].

MDSimAid automates the optimization of parameters for PME and MG, freeing users from the time consuming process of a manual search. Starting with published recommendations, we performed extensive empirical testing to determine which factors have the greatest impact on accuracy and runtime performance. Using these results, we have implemented heuristics that efficiently search the parameter space of PME and MG and return near optimal configurations.

We observe large improvements over published recommendations. For a force error tolerance of 1%, MDSimAid's recommendation for MG is up to 14 times faster or 4 times more accurate than the initial prediction. For a force error tolerance of 0.1%, MDSimAid's recommendation is up to 10 times faster or 17 times more accurate. We do not observe substantial improvements in the sequential runtime of PME (although we can optimize the accuracy 27-fold at a high cost). However, we are able to improve the parallel scaling of PME by reducing the amount of work spent in the FFT by the algorithm while not hurting the accuracy. In our tests, optimized PME can run up to twice as fast in parallel. For parallel runs, MDSimAid can find the optimal number of processors to run on, and optimize parameters related to parallel execution on that many processors.

An additional advantage of MDSimAid is that it can help users prepare their configuration files for the C++ MD packages NAMD 2.5 [18] or ProtoMol 2 [19]. Optimizations and input preparation for other MD software can easily be added. MDSimAid operates as a web portal [20]. Python source and examples may be obtained from the web page.

1.1 Recommender systems

The purpose of a recommender system is to rank a set of algorithms on particular problems according to some performance criteria. In scientific fields, recommender systems are often used to suggest optimal choices of software (PYTHIA-II [21], MyPYTHIA [22], SANS [23]), algorithms (SPIRAL [24]), and input configurations (PBCAID [25]). PBCAID, for example, optimizes the rotational position of a molecular system to reduce the size of the bounding box used in periodic MD simulations. Smaller bounding boxes provide substantial savings, but are time consuming to calculate by hand.

There are many approaches that recommender systems use for the optimization process: random searches, linear and nonlinear optimization, dynamic programming, tree searches, and machine learning techniques. The methodology typically includes off-line (*a priori*) and online (at run time) profiling of performance data and construction of performance models. We use off-line profiling of performance data to produce heuristics that guide the optimization at run time, where the initial values comes from analytical performance models.

2 Characteristics of fast electrostatics solvers

In this section, we briefly describe the fast electrostatics solvers Ewald, PME, and MG.

2.1 Ewald summation

A derivation and analysis of the Ewald summation method is found in references [1,2]. Ewald splits Eq. (1) into the sum of two rapidly converging sums:

$$\frac{1}{r} = \underbrace{\left(\frac{1}{r} - f_{\mathrm{smooth}}(r)\right)}_{\mathrm{Real-space}} + \underbrace{f_{\mathrm{smooth}}(r)}_{\mathrm{Reciprocal-space}} , \tag{3}$$

where $f_{\mathrm{smooth}}(r_{ij,\mathbf{m}} \equiv \mathbf{r}_j - \mathbf{r}_i + \mathbf{m}) = \frac{\mathrm{erf}(\alpha r_{ij,\mathbf{m}})}{r_{ij,\mathbf{m}}}$ is the softening function chosen by Ewald. The real-space sum is of the form:

$$\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \sum_{\mathbf{m}}{}' \left(\frac{1}{r_{ij,\mathbf{m}}} - \frac{\mathrm{erf}(\alpha r_{ij,\mathbf{m}})}{r_{ij,\mathbf{m}}}\right), \tag{4}$$

and the reciprocal-space sum:

$$\frac{1}{2\pi\varepsilon_0 L^3}\sum_{\mathbf{m}\neq 0}\frac{1}{\mathbf{m}^2}\exp\left(\frac{-\pi^2\mathbf{m}^2}{\alpha^2}\right)\hat{\rho}(\mathbf{m})\hat{\rho}(-\mathbf{m}), \tag{5}$$

where

$$\hat{\rho}(\mathbf{m}) = \sum_j q_j \exp(2\pi i \mathbf{m}\cdot\mathbf{r}_j). \tag{6}$$

The parameter α determines the convergence rates of the real- and reciprocal- space sums. Optimal choice of α leads to an $O(N^{3/2})$ algorithm [11, 15].

2.2 Particle Mesh Ewald

PME is similar to Ewald, but PME chooses the splitting parameter α such that the real-space part is $O(N)$, and interpolates the Fourier series, Eq. (5), onto a mesh, thus allowing the use of

$O(N \log N)$ FFT for the reciprocal-space part, Eq. (6):

$$\hat{\rho}(\mathbf{m}) \approx \sum_n \exp(2\pi i \mathbf{m} \cdot \mathbf{r}_n^h) \sum_j \phi_n(\mathbf{r}_j) q_j, \tag{7}$$

where $\phi(\mathbf{r})q$ does an adjoint interpolation (also called anterpolation) of the particle charges to the grid, and $\mathbf{r}_n^h$ are grid positions.

There are two approaches to solving the Ewald summation: as the solution of Poisson's equations in PBC, and as a large but finite array of copies of the simulation cell immersed in a dielectric medium. The former is used in the Lattice Gaussian Multigrid (LGM) method of [26]. The latter is used by the MG method we optimize. LGM uses the particle-mesh (PM) approach, but then transforms the problem in the mesh to the solution of an elliptic PDE, and uses an $O(N)$ iterative multigrid solver, achieving an overall $O(N)$ method. This is a high-accuracy scheme, unlike multigrid summation algorithms of [6, 8, 27–29]. LGM is expected to be more expensive and less scalable in parallel due to the iterative solver, and to the fewer levels of parallelism available in the method. LGM is typically slower than PME for a single processor, whereas MG is faster than PME for a single processor.

2.3 Multigrid summation

The MG summation splits the contributions to the electrostatic energy of Eq. (1) into a short-range part that is evaluated using a cutoff, and a smooth part. Switching functions are used to bring the energy function smoothly to zero at a cutoff point. The smooth part is approximated on a grid, by first anterpolating the particle charges. This process is repeated recursively, creating a hierarchy of grids, hence the name multigrid summation. In general, 2 to 4 grids are common. Once the grids are formed, the values of the forces are calculated. The steps of the algorithm are set within Figure

Algorithm 1 Pseudo-code of a recursive multi-grid scheme, cf. [30].

```
main:
  1. anterpolate particle charges to the finest charge grid(1) – step (1);
  2. call multiscale(maxLevel, level := 1);
  3. interpolate finest force grid(1) to particles – step (3);
  4. correct particle energies – step (4);
  5. compute forces and total energy
multiscale(maxLevel, level := $k$):
  1. if maxLevel = $k$ then
      (a) compute force values on coarsest grid(maxLevel) – step (2);
  2. otherwise
      (a) anterpolate charge grid($k$) to coarser charge grid($k+1$) – step (1);
      (b) call multiscale(maxLevel, $k+1$);
      (c) interpolate coarser force grid($k+1$) to finer force grid($k$) – step (3);
      (d) correct force grid($k$) – step (4)
```

1, starting from the lower left. Algorithm 1 gives an overview of the MG method, while Figure 1 shows schematically what is happening.

3 Optimization methodology

The focus of MDSimAid is to optimize PME and MG using Ewald as a benchmark. Starting parameter values for each method are generated using the desired accuracy and system size as input. Default values come from equations in the literature. The sequential optimization works in two stages by first obtaining the greatest accuracy and then relaxing the parameters to decrease the runtime while maintaining the desired accuracy. In the case where MDSimAid is unable to reach the desired accuracy, the focus becomes minimizing the cost metric, cf. Eq. (13). An overview of MDSimAid is given in Algorithm 2, with details following in Sections 3.1 – 3.3.

Algorithm 2 MDSimAid

1. After generating the initial configuration, perform optimizations on each parameter in the designated order to decrease the error in the simulation. At each step, store results along with an overall performance metric.

    (a) Perform a local search by varying r_c and h using small increments. (MG)

    (b) h: Using the most accurate test configuration from Step (1a) decrease the grid-size, h, until the accuracy is approximately constant. (MG)

    (c) r_c: Using the most accurate test configuration from Step (1b) for MG and from Step (1a) for PME, increase the softening distance, r_c, until the error is minimized. (MG and PME)

    (d) a: If the most accurate test configuration from Step (1c) is not within the error tolerance, increase the PME accuracy parameter, a, by a factor of 10 to decrease the error. (PME)

    (e) l: Using the five smallest values of h found in step (1c), increase the number of levels, l, until there are an insufficient number of grid points for an additional level of interpolation. (MG)

2. Having determined the best accuracy possible to this point, perform optimizations on each parameter in the designated order to decrease the runtime of the simulation.

    (a) l, h: For the set of configurations that satisfy the error tolerance, try to increase l by 1 and double h. By doubling h, we reduce the runtime and the accuracy; we compensate the accuracy loss by increasing l. (MG)

    (b) r_c: Decrease r_c until the error exceeds the desired error tolerance in order to decrease the runtime. Do this for up to five configurations within the error tolerance. (MG and PME)

3. Choose parameters that best meet user constraints while minimizing the cost metric and create necessary output files.

3.1 Optimization of MG

MG has five parameters: (*i*) the cutoff, r_c, used at the particle level; (*ii*) the grid size, h, at the finest grid; (*iii*) the number of grid levels, l; (*iv*) the smoothness of the switching function, k, (C^1, C^2, etc.); and (*v*) the interpolation order, p, from one level to the next.

MDSimAid starts MG runtime tests by evaluating an analytical model to determine initial values. The authors of [8] suggest balancing the work between the short-range and the smooth parts. For an approximation of order p to the smooth part of the force, the suggested values are given by:

$$h = \left(1+\frac{4}{p}\right)^{1/6}\left(\frac{64}{7}\theta\right)^{1/6} h_*, \tag{8}$$

$$r_c = \left(h^p h_*^2 \frac{C_p}{\varepsilon}\right)^{1/(p+2)}, \tag{9}$$

where h_* is the distance between two nearest neighbors; θ is the ratio of the cost of calculating a pairwise interaction between grid points to that of calculating a pairwise interaction between particles; ε is the desired accuracy; and C_p is a constant determined at run time.

Our simulations have shown that h is generally between 1 and 5 Å. Such spacing is suggested by Eqs. (8) and (9). Improvements in accuracy are obtained by decreasing h, increasing r_c (up to L), and through an increase in the number of levels, l. MG's advantage is the ability to interpolate to and from coarse and fine grids to get similar accuracy for less cost. Using additional levels is a major focus of MDSimAid's recommendations for MG. In theory, one could also increase p to improve the accuracy; however, our tests showed that this is a very costly operation, and unnecessary for the relatively low orders of accuracy obtained in periodic boundary conditions and that are needed for MD. This is also true for the switching function parameter, k.

Figure 2(a) illustrates the effect of increasing l on accuracy. The error is decreasing for each system, as expected. However, Figure 2(b) demonstrates that the increased accuracy comes with a price, particularly for small systems. Eventually, the overhead of the interpolation outweighs the computational savings in computing the summation.

We generate the default MG parameters and in Step (1a) perform a local search around r_c and h using small increments to look for quick improvements. The configuration from the local search that has the smallest error is used to continue the accuracy optimizations. In Step (1b), h is modified with larger increments until further changes in the error are negligible. For MG, h plays a significant role affecting both error and runtime. Step (1c) takes the configuration from the previous step and increases r_c until the error is minimized. In Step (1e), l is increased while keeping the finest grid size the same. Generally, an increase in l causes the greatest reduction of error when compared to changes in h or r_c, as is shown in section 4.4. However, we first increase h and r_c to allow the increase of l. If either h or r_c is too small, there will not be enough support for the interpolation to coarser grids. The order and switching function parameters for MG are not changed by MDSimAid for the reasons stated above. For these parameters, we use the default values of $p = 4$ and $k = C^2$.

For the second stage, Step (2a) increases l and h simultaneously. Making the grid coarser would normally increase the error, but increasing the levels makes up for the accuracy while also improving time significantly. Step (2b) reduces r_c for as long as the error is within the tolerance.

3.2 Optimization of Ewald

Fincham [17] and other researchers [14, 31] recommend that α vary with N such that

$$\alpha = c\sqrt{\pi}\left(\frac{N}{V^2}\right)^{1/6}, \tag{10}$$

where V is the volume of the system and c is the ratio of execution time of the real part to that of the reciprocal. Since c may vary from one platform to another, this ratio is computed by ProtoMol 2 at run time.

In addition to α, the direct space cutoff distance r_c, controls accuracy and CPU time in Ewald. The relationship between α and r_c is given by

$$r_c = \frac{\sqrt{-\ln \varepsilon}}{\alpha}, \tag{11}$$

where ε is the desired accuracy. A typical value for ε is 10^{-5}. Large α yields a small cutoff distance, faster execution time, and lower accuracy. Since Ewald is easily optimized and highly accurate, MDSimAid uses Ewald to determine the error in PME and MG.

3.3 Optimization of PME

PME has three method parameters: the first, α, determines the amount of work that is done in real and reciprocal space; the second is the cutoff r_c used to evaluate Eq. (4); the third is h, the grid point separation used by the FFT. The accuracy and the cost are proportional to r_c and inversely proportional to h.

The α and r_c parameters have the same qualitative effects in PME as in Ewald. Petersen [14] showed that the optimal α is approximately $N^{1/3}$, whereas the optimal r_c is proportional to $\log_2^{1/2} N$.

In particular, once α is determined (based on N), r_c is estimated as follows:

$$\frac{\text{erfc}(\alpha r_c)}{r_c} = \varepsilon, \tag{12}$$

where ε is the desired accuracy.

In PME, the value of h has a less significant impact on the accuracy and runtime than in MG. Default values of h are generated and are not changed in the optimization. We use cubic B-spline interpolation for PME.

The initial parameters are calculated from the analytical models, while, at the same time, an appropriate h based on the system size is determined. In Step (1c), r_c is reduced. MDSimAid then checks to see if the desired accuracy has been reached. If not, a is reduced by a factor of 10 in Step (1d). This step has the greatest impact on the accuracy. The fastest configuration found up to that point that is also within the desired accuracy is chosen for Step (2b), where r_c may be reduced.

3.4 Parallel optimization

MDSimAid can also optimize parallel runs. Our approach is to first optimize the methods sequentially using Algorithm 2, and then apply techniques to make the most efficient use of the parallel resources. For our tests, we assumed that minimizing runtime was the most important metric. However, MDSimAid may be easily modified to use different metrics or add constraints such as minimum acceptable parallel efficiency.

Users often desire to run simulations with the maximum number of processors available. This is not necessarily the best use of resources. As the number of processors increases, so too does the communication and bookkeeping. Eventually, the overhead dominates the work done by each processor and the runtime actually increases. Given a maximum number of processors P available to the user, we perform a modified binary search to determine the best runtime using at most P processors. A short simulation is run with 1, P, and $\frac{P-1}{2}$ processors and each runtime is recorded. We then choose the interval containing the shortest time and repeat this process. This results in approximately $\log_2 P$ simulations, from which we choose the fastest. Because of the variable nature of parallel processing, subsequent simulations may suggest differing numbers of processors. However, this heuristic often gives near optimal values without having to do an exhaustive search.

In addition to general improvements, we are able to boost the scalability of PME. Generally, the FFT used by PME does not scale well in parallel. However, we found that if we decrease the accuracy of the FFT ;method and simultaneously increase the interpolation order, we can improve the scalability of PME while maintaining similar accuracy. As seen in Figure 4(a), this optimization makes PME run nearly twice as fast in parallel. Note that this optimization does not work for the sequential version of PME due to the extra computation involved.

Finally, we are able to tune low-level, implementation dependent, parameters related to parallel runs, such as message granularity and load-balancing schemes. Based on the number of processors available for the parallel simulation, starting values are chosen and improved upon to obtain modest runtime improvements. For example, ProtoMol 2 has 3 options directly related to parallel execution: `parallelMode`, `parallelPipe`, and `maxPackages`. These parameters control load-balancing, communication frequency and packet size. Figures 5(a) and 5(b) show the effect of `parallelPipe` and `maxPackages` on runtime for two different load-balancing schemes.

The effect of optimizing the parallel parameters in ProtoMol 2 is not nearly as significant as adjusting for speedup through additional processors. However, tests show that a 10%-20% improvement may be achieved.

4 Tests

First, we test that the recommendations of MDSimAid for sequential PME and MG have reasonable scalability and performance. We find that for moderate accuracy (defined below), MG is able to run as fast or faster than PME for all our tests. PME is generally faster if very high accuracy is desired. Since MD makes many approximations, smoothness of computed forces is more important than accuracy.

Second, we confirm that MDSimAid is able to use parallel resources effectively by quickly finding the optimal number of processors for PME and MG with two implementations: one using atom decomposition (AD) and another using force decomposition (FD), cf. [6, 32]. For PME, a spatial decomposition (SD) implementation in NAMD 2.5 is also benchmarked. We find that the force-decomposition MG is more scalable than all three implementations of PME tested.

Third, we evaluate the effectiveness of MDSimAid's optimizations. We find that MDSimAid significantly improves runtime and accuracy of MG. We are also able to improve the parallel scalability of PME by moving work from the FFT step to the interpolation step as mentioned above. Once we have found the best parallel runtime, we are able to improve it by about 10-20% by optimizing low-level parameters related to the parallel implementations.

Finally, we evaluate the stability of MD simulations of solvated proteins produced by the recommendations of MDSimAid, both for MG and PME. We find that both methods produce stable and accurate results. Although this is not an exhaustive evaluation, it gives assurance that the optimizations are not too aggressive. We describe our methods, and then detailed results.

4.1 Methods and systems

MDSimAid's rules were obtained by testing flexible TIP3P [33] water boxes and solvated protein systems from 1,000 to 400,000 atoms. The water boxes were generated using VMD [34]. For each water box, 1000 minimization steps were run with NAMD 2.5. Then, starting from a temperature of 0 K, 1000 steps were run between increments of 25 K until a temperature of 300 K was reached. For equilibration, additional 8000 steps were run at 300 K. All these equilibration runs used a Verlet/leapfrog integrator with time step of 1 fs, and cutoff for Lennard–Jones and Coulomb of 12 Å. The switching distance for a C^2 switching function for Lennard–Jones was 8 Å. To run the experiments comparing MG and PME in ProtoMol 2 and PME in NAMD 2.5, a short cutoff of Lennard–Jones of 6 Å and a switching distance of 3 Å was used for all methods. The reason is that NAMD 2.5 only reports runtime for the whole MD integration and force calculation. Thus, we tried to minimize the time spent in non-Coulomb calculations. For PME, the separation between grid points was set to 1 Å, a standard practice. MG was run with 2 grid levels, a C^2 switching function, cubic interpolation, a smoothing distance of 12 Å in the finest grid, and between 1000 and 3000 grid points in the finest grid. These tests use a multiple time stepping r-RESPA scheme where every 1 fs all bonded forces and the direct and correction parts of MG or PME are evaluated, and every 4 fs the smooth or reciprocal parts of MG or PME are evaluated. Both ProtoMol 2 and NAMD 2.5 are compiled without MPI for uni-processor runs.

We also use solvated protein systems. Apolipoprotein A-I (apoA-I) is the primary protein constituent of high density lipoprotein (HDL), which circulates in the bloodstream, extracts cholesterol from body tissues and transports it to the liver for excretion or recycling [35, 36]. NAMD 2.5 uses a solvated structure of apoA-I with 92,224 atoms as a performance benchmark. We use the same benchmark to test MDSimAid. For the apoA-I simulations, the same parameters used by the

NAMD 2.5 performance benchmark were used: a switching distance of 8 Å, a cutoff of 10.5 Å for the Lennard–Jones and direct part of MG, and r-RESPA with all bonded forces and short range Lennard–Jones and Coulomb evaluated every 1 fs, and the smooth or reciprocal part of MG or PME evaluated additionally every 4 fs. Three runs were performed for each data point in the results. The error bars were smaller than the symbols. Both ProtoMol 2 and NAMD 2.5 are compiled for using the MPI library with Myrinet support. We noted that the Myrinet MPI version of NAMD 2.5 consumes significantly more memory than the uni-processor version, and also than ProtoMol 2. All of these simulations were run under PBC using the CHARMM 28a2 [37] force field.

We also ran a long vacuum simulation using crambin, a small protein with PDB id 1EJG. We solvated it in flexible TIP3P water for a total of 2,277 atoms. We minimized and equilibrated it using NAMD 2.5 using an identical procedure to the apoA-I. Then, we optimized MG and PME with MDSimAid, and performed a simulation for 500 ps using a triple time stepping r-RESPA, with innermost time step of 0.75 fs for bonds, angles, Lennard–Jones and direct part of MG; 1.5 fs time step for impropers, dihedrals, and the correction for MG; and 3 fs time step for the smooth part of MG. We remove every few steps the linear and angular momentum of the center of mass of the system. This is necessary in vacuum simulations in general to prevent movement of the molecule. We confirmed this is necessary by running simulations using plain cutoffs. Thermodynamical, structural, and dynamic metrics tested for MG and PME match.

We ran these tests in an Atipa Linux cluster (`http://iss.cse.nd.edu`) with 44 dual Xeon 2.4 GHz processors with Myrinet interconnect. There is a total of 90 Gigabytes of memory. This cluster runs Linux Red Hat Linux 8.0 3.2-7, kernel 2.4.18-18.8.0smp. ProtoMol 2 2.0.2 was configured using `‘‘configure --with-gcc-mpich’’` and uses Myrinet MPI libraries. NAMD 2.5 was configured using `‘‘config tcl fftw plugins Linux-i686-MPI’’`. It uses the parallel run-time system

`Charm++`. This was built using ``` ``build charm++ mpi-linux gm2 -O -DCMK_OPTIMIZE=1'' ``` to include Myrinet MPI libraries and to optimize the library.

MDSimAid uses three different values for analyzing the effectiveness of each configuration, or set of parameters. The first is runtime t, which is simply the amount of time used by ProtoMol 2 to complete one simulation step. The second, average error $rFavg$, comes from ProtoMol 2's built-in force comparator. Finally, the cost metric, M_e, is defined as

$$M_e = \frac{t}{|\log_{10}(rFavg)|}, \tag{13}$$

and is a simple way to combine runtime and error in a single value. M_e is the time it takes to achieve a ten-fold improvement in the error. A lower M_e means either that the runtime cost to attain the desired accuracy was lower, or that a smaller error was attained with virtually the same runtime cost.

ProtoMol 2 computes the relative error in force and potential energy by using a specified method as a reference. In the case of MDSimAid, the reference method is Ewald. Errors are computed by ProtoMol 2 comparing Ewald to MG or Ewald to PME. The metric for relative force error is defined as follows:

$$rFavg \quad = \quad \frac{\sum_i \sqrt{\frac{||\mathbf{F}_i - \tilde{\mathbf{F}}_i||^2}{m_i}}}{\sum_i \sqrt{\frac{||\mathbf{F}_i||^2}{m_i}}}, \tag{14}$$

where m_i is the mass of atom i; $\mathbf{F}$ is the force of the reference algorithm; and $\tilde{\mathbf{F}}$ is the force of the algorithm being tested.

4.2 Sequential PME and MG

Figure 3 shows the sequential performance of MDSimAid-optimized MG and PME in ProtoMol 2 and PME in NAMD 2.5. Note that our implementations are reasonably optimized as compared to NAMD 2.5's. For error tolerances of 10^{-2} and 10^{-3}, MG performs better than PME in all cases. For error tolerances smaller than 10^{-3}, MG is unable to match PME. Our implementation of MG in PBC is limited to accuracies of 10^{-3} at a reasonable cost. These errors do not hinder the ability of MD simulations to reproduce dynamical and structural features. The smoothness of the forces computed with MG makes up for the moderate accuracy.

4.3 Parallel MG and PME

Figures 4(a) and 4(b) show the best parallel runs found by MDSimAid for atom decomposition implementations of PME and MG in ProtoMol 2 for systems of up to 200,000 atoms, on 26 processors. As explained above, there are three steps: sequential optimization; finding the best runtime using a binary search on the number of processors; and fine tuning low-level parameters of the parallel implementation. Note the lines labeled "PME OPT - AD". These show the optimization of PME in parallel achieved by MDSimAid.

Figures 6(a) and 6(b) show the MDSimAid-optimized parallel runs of AD and FD PME and MG in ProtoMol 2 and spatial decomposition PME in NAMD 2.5 for apoA-I. These illustrate the ability of MDSimAid to optimize a run in a specific number of processors. They also show the greater scalability of FD MG over PME for 64 or more processors.

4.4 Evaluation of MDSimAid's effectiveness

Figures 7(a) and 7(b) compare MDSimAid's recommendations for MG against those suggested by Eqs. (8)–(9). The tests were run with ProtoMol 2 using MG on molecular systems of sizes 8,000 to 80,000 atoms. Results are presented showing both $rFavg$ and the cost metric M_e. As the system size increases, the recommender is more efficient at optimizing M_e.

Figures 8(a) and 8(b) show the error reduction of each optimization step for MG with respect to the previous one, as well as the overall error reduction for $\varepsilon = 10^{-2}$, and $\varepsilon = 10^{-3}$. The error bar is computed over 9 runs with different target accuracies. Similarly, Figures 9(a) and 9(b) show the speedup of each optimization step for MG with respect to the previous one, as well as the overall speedup and error bars for the corresponding accuracies in Figures 8(a) and 8(b).

Figure 10 shows that MDSimAid only moderately improves the accuracy of PME up to $\varepsilon = 10^{-5}$ with little effect on the runtime. The same figure also demonstrates that for accuracies near $\varepsilon = 10^{-6}$, MDSimAid produces an average 27 fold improvement when compared to Eq. (12), but at a large 10-fold increase in running time. Since higher accuracy PME is not a requirement for MD simulations, we concentrate on improving the parallel performance of PME instead.

These tests involve 8 water systems containing 1,029 to 80,001 atoms. Each system was tested with 18 error tolerances. For MG, we used values between 2.5×10^{-4} and 5×10^{-2}. Since PME can attain higher accuracy, we used values between 7.5×10^{-7} and 5×10^{-4}. These values were chosen to go beyond the accuracy capabilities of each method in order to fully test MDSimAid. With a sufficient number of atoms, MG and PME can attain accuracies of 10^{-3} and 10^{-6} respectively.

For MG, optimizing h and l have the highest impact on both accuracy and runtime (Steps (1b), (1e), (2a)). Whereas, for PME, optimizing r_c has the highest impact on accuracy.

4.5 Stability of Simulations

Here we illustrate that the recommended parameters for MG produce stable MD simulations. Neither linear momentum nor angular momentum are theoretically preserved by MG. The introduction of a gridding of space perturbs the value of the potential energy function so that it is not invariant to rigid body rotation or translation. The momenta fluctuate around a constant value and do not suffer from overheating or freezing.

To validate that the momentum oscillates around a constant value with MG, we use the results of the crambin simulation described above. Figures 11(a) and 11(b) show the total energy and linear momentum of the system. We discard the first 150 ps of simulation and plot the next 350 ps. It can be seen that despite the long time steps, the energy and momentum are conserved in practice.

5 Discussion

MDSimAid has proven to be a practical tool for generating efficient parameters for PME and MG, thereby making them more accessible. This is especially true for parallel runs. Even so, there are many features of MDSimAid that could be improved.

The methodology of MDSimAid could be extended by making it into an adaptive application that improves future recommendations. The run-time optimizations could be performed in a cluster or computational grid, and the search algorithms could be enhanced.

MDSimAid could be extended to optimize more fast electrostatic solvers, such as FMM and others, cf. [5, 26, 27, 38–40]. Optimization of fast electrostatics in combination with multiple time stepping integrators is another area for further development, cf. [41].

MDSimAid has the ability to interface with well known MD simulators, such as NAMD [42], CHARMM [37], AMBER [43], and GROMACS [44,45]. Although our tool will be somewhat sensitive to the particular implementation of the method, our approach is reasonable and can adapt to different optimizations because it is dynamic: it does profiling, at run time, of the performance and accuracy of the methods, on a particular computer system and for a specific problem. Also, we show that our implementation of PME is reasonably efficient when compared to a highly optimized PME in NAMD 2.5. Finally, to the best of our knowledge, no other MD package has implemented the MG summation described here. In [6] we describe the extension to periodic boundary conditions tested here.

5.1 Acknowledgments

This work was partially supported by an NSF Career Award ACI-0135195. Computations were performed in part through a Beowulf cluster supported by NSF grant DMR-0079647 and also by an Equipment Renovation grant through the University of Notre Dame. Michael Crocker was supported by an Interdisciplinary Center for the Study of Biocomplexity at Notre Dame Undergraduate Research scholarship and NSF REU ACI-0135195. Scott Hampton was partially supported through an Arthur J. Schmitt fellowship.

References

[1] S. W. de Leeuw, J. W. Perram, and E. R. Smith, *Proc. R. Soc. Lond. A*, **373**, 27–56 (1980).

[2] P. Ewald, *Ann. Phys.*, **64**, 253–287 (1921).

[3] T. Darden, D. York, and L. Pedersen, *J. Chem. Phys.*, **98**(12), 10089–10092 (1993).

[4] U. Essmann, L. Perera, and M. L. Berkowitz, *J. Chem. Phys.*, **103**(19), 8577–8593 (1995).

[5] L. Greengard and V. Rokhlin, *J. Comput. Phys.*, **73**, 325–348 (1987).

[6] J. A. Izaguirre and T. Matthey Submitted to *Journal of Parallel and Distributed Computing.* Manuscript available at `http://www.nd.edu/~izaguirr/papers/MaIzOx_JPDC.pdf`, (2004).

[7] A. Brandt, J. Bernholc, and K. Binder, Eds., *Multiscale Computational Methods in Chemistry and Physics*, Vol. 177 of *NATO Science Series: Series III Computer and Systems Sciences*, IOS Press, Amsterdam, Netherlands, 2001.

[8] R. D. Skeel, I. Tezcan, and D. J. Hardy, *J. Comp. Chem.*, **23**(6), 673–684 (2002).

[9] E. L. Pollock and J. Glosli, *Comput. Phys. Commun.*, **95**, 93–110 (1996).

[10] M. Kawata and M. Mikami, *J. Comp. Chem.*, **21**(3), 201–217 (2000).

[11] A. Y. Toukmaji and J. A. Board, *Comput. Phys. Commun.*, **95**, 73–92 (1996).

[12] P. F. Batcho, D. A. Case, and T. Schlick, *J. Chem. Phys.*, **115**(9), 4003–4018 (2001).

[13] R. Zhou, E. Harder, H. Xu, and B. J. Berne, *J. Chem. Phys.*, **115**(5), 2348–2358 (2001).

[14] H. G. Petersen, *J. Chem. Phys.*, **103**(3668–3679) (1995).

[15] M. Deserno and C. Holm, *J. Chem. Phys.*, **109**(18), 7678–7693 (1998).

[16] C. Sagui and T. A. Darden, *Ann. Rev. Biophys. Biomol. Struct.*, **28**, 155–179 (1999).

[17] D. Fincham, *Mol. Sim.*, **13**, 1–9 (1994).

[18] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, *J. Comput. Phys.*, **151**, 283–312 (1999).

[19] T. Matthey, T. Cickovski, S. S. Hampton, A. Ko, Q. Ma, M. Nyerges, T. Raeder, T. Slabach, and J. A. Izaguirre, *ACM Trans. Math. Softw.*, **30**(3), 237–265 (2004).

[20] MDSimAid, Molecular Dynamics Simulation Aid `http://mdsimaid.cse.nd.edu/`, 2001.

[21] E. N. Houstis, A. C. Catlin, J. R. Rice, V. S. Verykios, N. Ramakrishnan, and C. E. Houstis, *ACM Trans. Math. Softw.*, (2), 227–253 (2000).

[22] E. N. Houstis, A. C. Catlin, J. R. Rice, N. Ramakrishnan, and V. S. Verykios, *Concurrency and Computation: Practice and Experience*, **14**(13–14), 1481–1505 (2002).

[23] J. Dongarra and V. Eijkhout, *Int. J. of High Perf. Comp. App.*, **17**(2), 125–131 (2003).

[24] M. Püschel, B. Singer, J. Xiong, J. Moura, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson, *Int. J. of High Perf. Comp. App.*, **18**(1), 21–45 (2004).

[25] PBCAID, Pbcaid `http://monod.biomath.nyu.edu/index/software.html`, 2002.

[26] C. Sagui and T. Darden, *J. Chem. Phys.*, **114**(15), 6578–6591 (2001).

[27] L. Y. Zaslavsky and T. Schlick, *Appl. Math. Comput.*, **97**, 237–250 (1998).

[28] A. Brandt and A. A. Lubrecht, *J. Comput. Phys.*, **90**, 348–370 (1990).

[29] B. Sandak, *J. Comp. Chem.*, **22**(7), 717–731 (2001).

[30] T. Matthey *Framework Design, Parallelization and Force Computation in Molecular Dynamics* PhD thesis, University of Bergen, Bergen, Norway, (2002).

[31] Z. Wang and C. Holm, *J. Chem. Phys.*, **115**(14), 6351–6359 (2001).

[32] S. Plimpton and B. Hendrickson, *J. Comp. Chem.*, **17**(3), 326 (1996).

[33] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, *J. Chem. Phys.*, **79**, 926–935 (1983).

[34] W. F. Humphrey, A. Dalke, and K. Schulten, *J. Mol. Graphics*, **14**, 33–38 (1996).

[35] J. H. Wald, E. S. Krul, and A. Jonas, *J. Biol. Chem.*, **265**, 20037–20043 (1990).

[36] A. Jonas, K. E. Kezdy, and J. H. Wald, *J. Biol. Chem.*, **264**, 4818–4824 (1989).

[37] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comp. Chem.*, **4**(2), 187–217 (1983).

[38] D. York and W. T. Yang, *J. Chem. Phys.*, **101**, 3298–3300 (1994).

[39] P. Bode and J. P. Ostriker, *Astrophysical J. Supplement Series*, **145**(1), 1–13 (2003).

[40] Z. H. Duan and R. Krasny, *J. Comp. Chem.*, **22**(2), 184–195 (2001).

[41] M. Tuckerman and B. Berne, *J. Chem. Phys.*, **94**(10), 6811–6815 (1991).

[42] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, *J. Comput. Phys.*, **151**, 283–312 (1999).

[43] P. K. Weiner and P. A. Kollman, *J. Comp. Chem.*, **2**, 287–303 (1981).

[44] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen, *Comput. Phys. Commun.*, **91**, 43–56 (1995).

[45] E. Lindahl, B. Hess, and D. van der Spoel, *J. Mol. Model.*, **7**(8), 306–317 (2001).

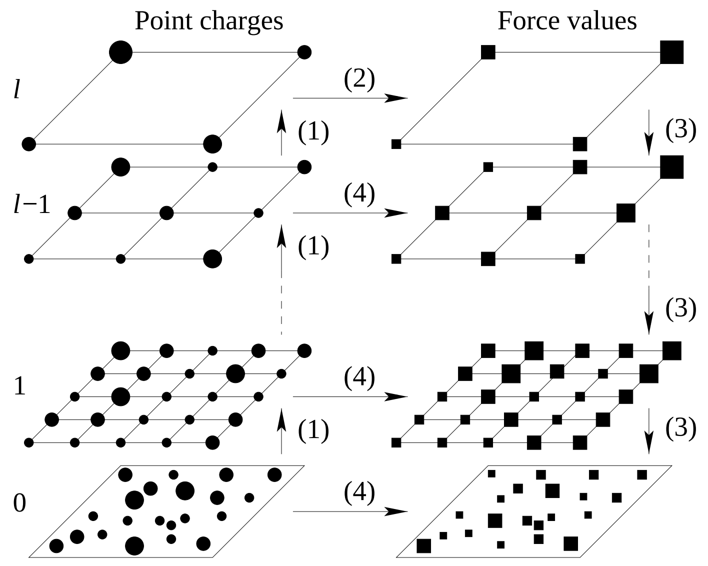
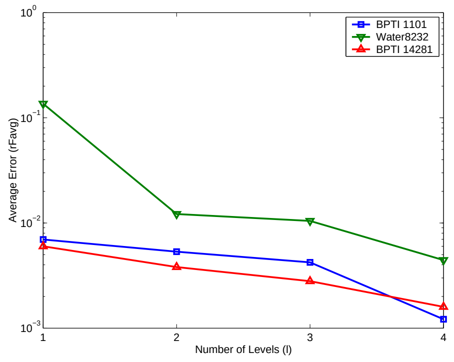
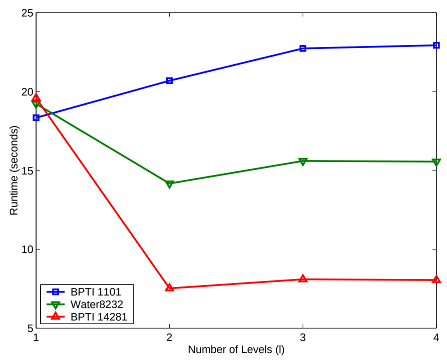


Figure 1: A pictorial view of the MG algorithm, cf. [30]: (1) Aggregate to coarser grids; (2) Compute potential induced by the coarsest grid; (3) Interpolate potential values from coarser grids; (4) Local corrections.





(a) (b)

Figure 2: The effect of number of levels, l, on (a) the average error, and on (b) the runtime. There are 32 grid points per dimension in the finest level grid.

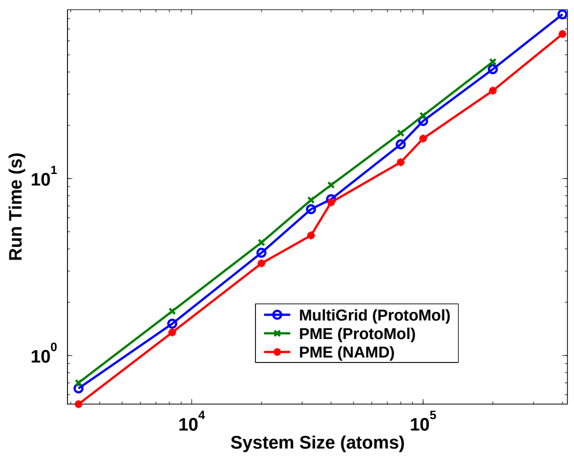
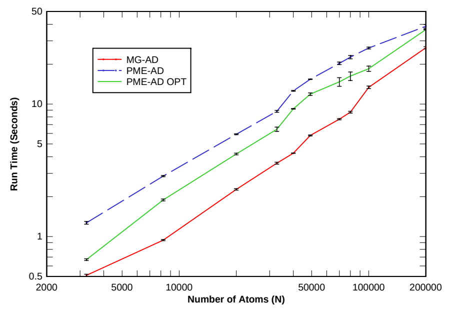
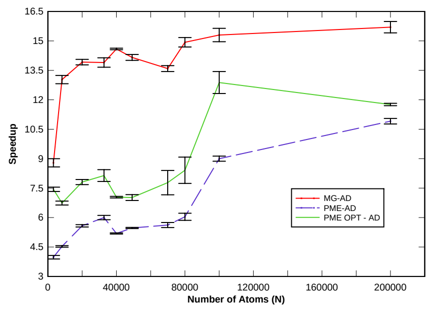


Figure 3: Single processor runtime for 8 MD steps for N-body solvers implemented in PROTOMOL 2 and NAMD 2.5 with relative force error of order 10^{-3}. Runs performed on Linux cluster with 44 dual-processor Xeon 2.4GHz with Myrinet interconnect.

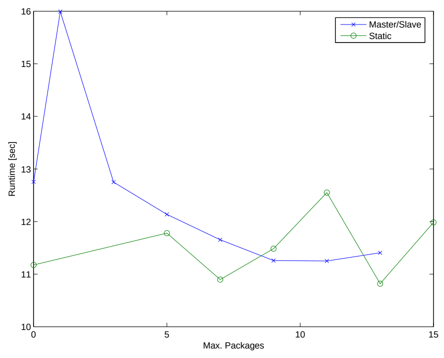


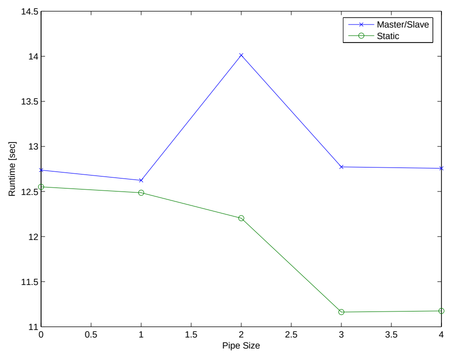


(a)

(b)

Figure 4: Comparison of best parallel runs for atom decomposition MG, PME, and optimized PME (PME OPT) in ProtoMol 2. The optimizations are performed by MDSimAid. At most 26 processors of a 44 dual node Xeon 2.4GHz clusters are used. (a) Runtime in seconds; (b) Parallel speedup. Error bars obtained from 3 runs for each data point.

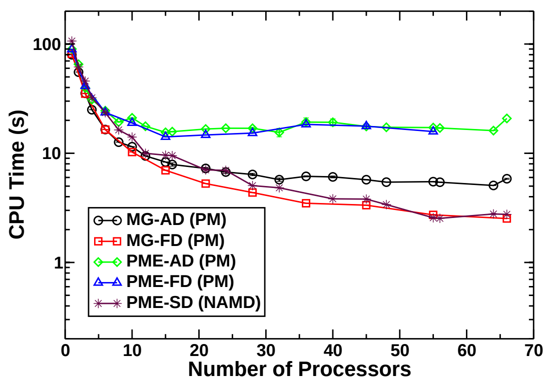




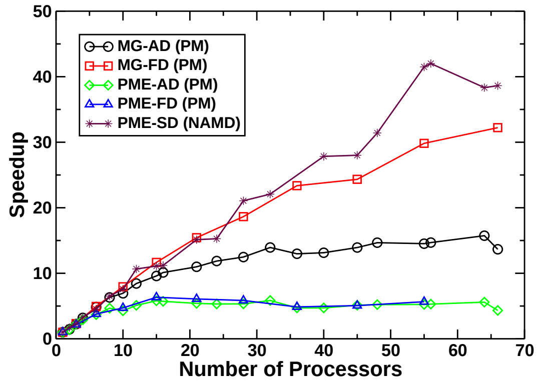
(a)

(b)

Figure 5: Comparison of ProtoMol 2 MPI parameters for master-slave and static load balancing. Both are for ProtoMol 2 MPI running on 16 processors using MG on a 44 dual node Xeon 2.4GHz cluster. (a) Maximum Packages Parameter (b) Parallel Pipe Parameter

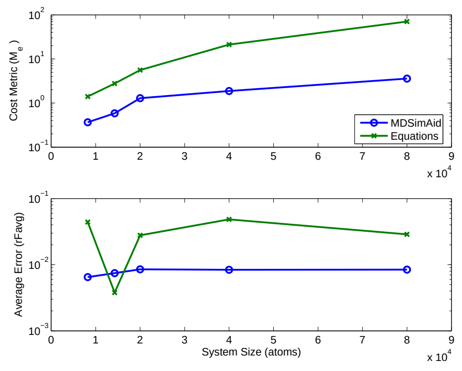


(a)

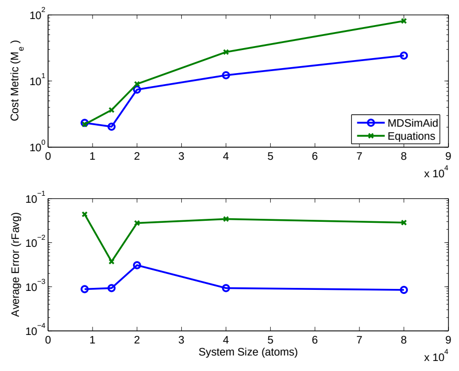


(b)

Figure 6: Comparison of MDSimAid-optimized parallel runs for different numbers of processors. We compare using atom (AD) or force decomposition (FD) for MG and PME in ProtoMol 2 as well as spatial decomposition (SD) in NAMD 2.5. Data is for solvated apoA-I with 92,224 atoms. Runs performed on a Linux cluster of 44 dual-processor Xeon 2.4 GHz with Myrinet interconnection network. (a) Runtime in seconds; (b) Parallel speedup.

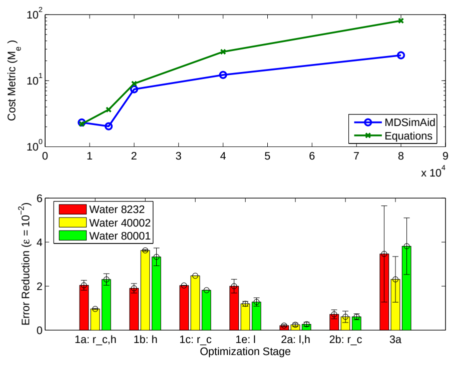


(a)

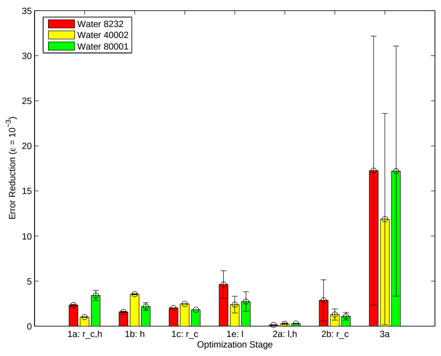


(b)

Figure 7: Recommendation of analytic methods vs. MDSimAid for MG with (a) $\varepsilon = 10^{-2}$, and (b) $\varepsilon = 10^{-3}$. The top graph displays the cost metric, M_e, while the bottom graph shows achieved accuracy.

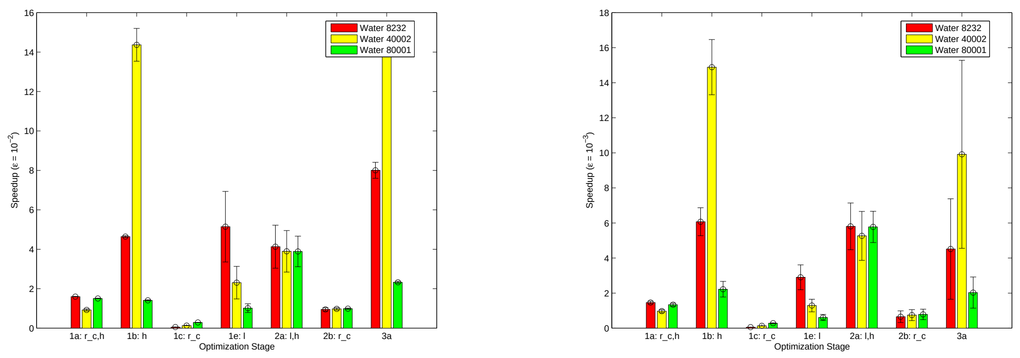


(a)



(b)

Figure 8: Error reduction relative to the previous step for MG. The last column is overall reduction from the initial equations. (a) shows values for $\varepsilon \geq 10^{-2}$, and (b) for $\varepsilon \geq 10^{-3}$.



(a) (b)

Figure 9: Runtime reduction relative to the previous step for MG. The last column is overall reduction from the initial equations. (a) shows values for $\varepsilon \geq 10^{-2}$, and (b) for $\varepsilon \geq 10^{-3}$.

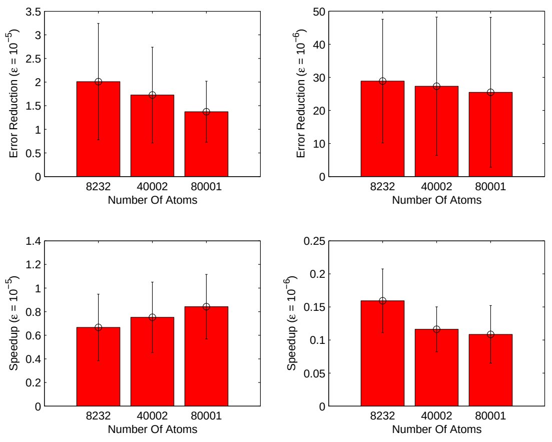
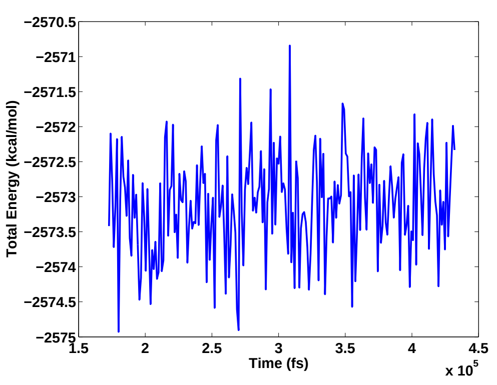
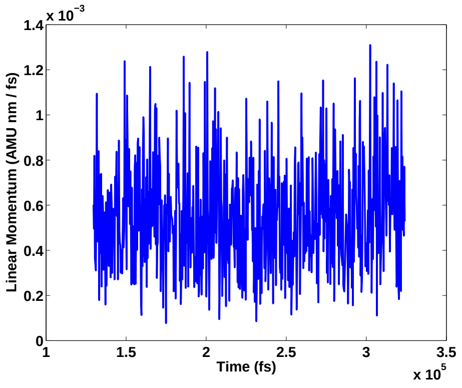


Figure 10: Average error and runtime optimization improvements for PME with $\varepsilon \geq 10^{-5}$, and $\varepsilon \geq 10^{-6}$ for 3 different TIP3P water systems.



(a)



(b)

Figure 11: (a) Total energy and (b) linear momentum for solvated crambin in vacuum with spherical boundary conditions using MultiGrid summation.