JCTC Journal of Chemical Theory and Computation

Efficient, Regularized, and Scalable Algorithms for Multiscale Coarse-Graining

Lanyuan Lu,[†] Sergei Izvekov,[†] Avisek Das,[‡] Hans C. Andersen,[‡] and Gregory A. Voth*,[†]

Center for Biophysical Modeling and Simulation and Department of Chemistry, University of Utah, Salt Lake City, Utah 84112-0850, and Department of Chemistry, Stanford University, Stanford, California 94305

Received December 2, 2009

Abstract: The multiscale coarse-graining (MS-CG) method obtains CG interactions from atomistic configurations, as demonstrated previously for a variety of soft matter and biological systems. In this article, recent advances in MS-CG algorithms are described, and a recently developed computer program MSCGFM for MS-CG calculations is introduced. The algorithms enhance the efficiency and stability of MS-CG computations, and these algorithms are incorporated into the MSCGFM program. As a result of these efforts, MS-CG calculations on large scale systems such as peptide and proteins can become tractable, and the numerical stability of solutions for ill-posed MS-CG problems can be regularized efficiently. Various parallelization strategies are also discussed.

1. Introduction

Molecular dynamics (MD) simulations with conventional empirical force fields such as Charmm,[1,2] Amber,[3] OPLS,[4] and Gromos[5] are widely used as a computational tool to study soft matter and biophysical systems. A standard all-atom force field typically contains parameters for molecular models with one point mass representing each atom or heavy atom. In typical atomistic MD simulations, observable time scales range from picoseconds to a few microseconds, and tractable system sizes are on the scale of nanometers. However, due to the temporal and spatial limitations of simulations based on all-atom resolution models, many processes in biology are not able to be modeled using available force fields.[6,7] In these cases, coarse-grained (CG) modeling is a powerful tool to study phenomena that involve larger time and length scales.[8] Here, we consider CG models that, like typical atomistic molecular models, regard the system as a set of point masses, called "CG sites". Each CG site corresponds to one or more of the atoms on the same molecule in an atomistic model. It is common to construct such CG models so that most molecules are represented by fewer sites than its number of atoms in the atomistic model. In some cases, some of the molecules, e.g., solvent molecules, have no sites at all. CG models have fewer degrees of freedom than the corresponding atomistic system and hence provide a lower resolution description of the molecular system.

There are numerous ways to develop CG models for different systems. Some approaches tune the parameters of the CG model to reproduce selected physical properties of particular systems. For example, in the MARTINI CG force field,[9] developed by Marrink and coworks and implemented for lipid bilayer[9] and membrane-protein systems,[10] parameters are chosen to reproduce the experimental partitioning free energy values for various components. Another category of methods is based on a strategy in which the low resolution CG model is developed using data from a high resolution model, e.g., an all-atom MD simulation. The advantage of this type of CG approach is that it is able to utilize existent and mature all-atom force fields. In such approaches, an ensemble of atomistic configurations is first generated by performing simulations of the high resolution model, and the low resolution model is then parametrized to reproduce certain properties of the high resolution ensemble. The key issue here is which properties to choose to link the models of different resolutions. In the inverse Monte Carlo[11] or

* Author to whom correspondence should be addressed. E-mail: voth@chem.utah.edu.

[†] University of Utah.

[‡] Stanford University.

10.1021/ct900643r
Published on Web 01/27/2010

iterative inverse Boltzmann[12] method, the radial distribution function (RDF) is usually chosen as the target for fitting in order to obtain the CG model parameters. The inverse Monte Carlo method has been applied to many systems. The reader is referred to the article of Murtola et al. for a recent review.[13] Some theoretical discussions of RDF matching can be found in the literature.[14−18] Higher order correlations, such as three-body correlation functions, are not retained by RDF fitting, and including higher order effects can become computationally intractable in the inverse Monte Carlo method.[17] Moreover, CG models that target the RDF do not in principle reproduce thermodynamic properties of the system.[19]

In the multiscale coarse-graining (MS-CG) method,[20−23] the key property that the CG model is constructed to fit is the many-body equilibrium probability distribution function of the positions of the CG sites in the atomistic system. A mapping operator is used to define the location of each CG site in an atomistic system as a function of the positions of the atoms. (For example, if a CG site corresponds to a set of atoms on a molecule, its location might be defined to be the center of mass or the center of geometry of that set of atoms.) The equilibrium distribution of atom positions in the atomistic system then implies an equilibrium distribution of the CG sites *in the atomistic system.* The MS-CG model that corresponds to the atomistic system is defined to be the model whose equilibrium distribution of CG sites is the same as in the atomistic system.

The potential energy function of the MS-CG model satisfies a variational principle. A numerical method for performing variational calculations to obtain approximate representations of that potential energy function has been developed. The method requires, as input data, a large set of configurations generated by computer simulations of the atomistic system at equilibrium. The method also requires a choice of basis functions to be used in the variational calculation. The choice of a linear basis set reduces the numerical work to standard but large scale problems in numerical matrix computation. The method has been successfully applied to a variety of liquids[21,24,25] and biological systems.[20,26−30]

The CG potential defined by the MS-CG method is in fact the many-body potential of mean force of the CG sites in the atomistic system at equilibrium. When a variational approximation for that potential and its gradients are used to simulate the CG model directly, the resulting forces acting on the sites can be regarded as renormalized forces that take into account the effect of the forces associated with the degrees of freedom that were eliminated in going from the atomistic to the CG system.

While the variational principle in MS-CG is strictly defined from the theoretical point of view, the numerical implementation of it is more challenging. The force-matching problem in MS-CG is a typical inverse problem in which one derives the CG model parameters from the data of atomistic forces, through minimizing a defined variational residual. In practice, the least-squares problem that arises thereafter can be written in the form of a linear matrix equation, by choosing appropriate basis functions to ensure the linearity. However, due to the nature of the molecular systems handled by the MS-CG approach, there are some numerical challenges to be overcome in developing highly efficient algorithms for solving the least-squares problem.

The computational aspects of the variational problem are challenging. The numerical problem can be formulated either as that of obtaining the exact solution of a very large set of inhomogeneous linear equations or as that of obtaining a least-squares solution of a much larger overdetermined set of such equations. Most algorithms for very large systems use the latter formulation.

The first challenge in solving the MS-CG least-squares problem arises from the huge dimension of the matrix for the very large complicated systems to which the MS-CG method is to be applied. For example, for problems of interest, the column dimension of the matrix varies from hundreds to tens of thousands. To make matters more challenging, the row dimension of the matrix is proportional to the product of the number of CG sites in an atomistic configuration and the number of atomistic configurations that are used as input data. In order to obtain good statistics for the CG potential and force functions, a large number of all-atom trajectory configurations are needed, which makes storing the matrix in the physical memory of a typical computer a formidable challenge. The whole matrix cannot be stored in memory, which makes it hard to apply many standard least-squares algorithms. Moreover, the enormous problem size can even make it impossible to store the whole matrix on a hard drive and implement some out-of-core algorithms for the matrix equation. The large size of the matrix equation highlights the need to develop "on the fly" algorithms and apply corresponding least-squares algorithms. The sparse nature of the matrix should also be utilized to enhance computation speed and reduce the storage requirements. For example, the matrix in the MS-CG equations typically contains less than 1% nonzero elements for many large scale systems. Therefore, it is clearly desirable to implement algorithms based on sparse matrix data structure for more efficient memory usage and faster computation.

Another challenge in the numerical implementation of the MS-CG equations arises from the fact that, for typical problems of interest, basis sets typically employed, and data sets typically used, the least-squares problem is numerically ill-posed, with the matrix being nearly singular. Especially for complex biomolecular systems, the ill-posed nature of the problem usually creates a degeneracy of solutions that can be affected by statistical noise in the input all-atom configuration data. One way to enhance the stability of the solution is to apply more robust least-squares solvers, such as those designed for ill-posed problems. Moreover, the so-called regularization method can be implemented, which can prevent the overfitting phenomenon and produce solutions with improved smoothness.

In this article, recent advances in the numerical solutions of the MS-CG equations are presented. These include algorithms for reading the all-atom MD trajectories and generating the matrix equation, as well as strategies for solving the least-squares problem. Data structure is closely related to the algorithms, and therefore it will also be discussed. Before these new algorithms were employed, our

original MS-CG program could perform calculations for CG models with only around 10 to 15 types of CG sites due to the limitations of storage space and computation speed. By contrast, the new algorithms in this paper extend the application scope of the MS-CG approach to complex biological systems such as proteins and membranes. All new algorithmic features described here are implemented in the MSCGFM program that has been recently developed for MS-CG calculations.

The structure of this paper is as follows: Section 2 briefly introduces the MS-CG methodology and discusses basis functions in the MS-CG least-squares problem. The choices for dense/sparse matrix data structures are then given in section 3, accompanied by a description of the neighbor list algorithm for generating the matrix equation. In section 4, different algorithms for solving the least-squares problem are discussed, and miscellaneous algorithms are described in section 5. Finally, the last two sections, 6 and 7, contain benchmark calculations and conclusions, respectively.

2. Multiscale Coarse-Graining Methodology

2.1. Summary of the MS-CG Calculations. The most recent state-of-the-art discussion of the MS-CG approach along with its theoretical background can be found elsewhere.[22,23] Here, we will discuss only enough to provide the background for the numerical considerations.

The goal of the MS-CG method is to develop a coarse-grained model for an atomistic system. The atomistic system is in general a molecular system whose configuration can be represented by $\mathbf{r}^n$, which is the collection of position vectors of the n atoms in the sytem. The total number of CG sites on all the molecules is N. The position of site I, denoted $\mathbf{R}_I$, is defined as $\mathbf{R}_I = \mathbf{M}_{\mathbf{R}I}(\mathbf{r}^n)$, where $\mathbf{M}_{\mathbf{R}I}(\mathbf{r}^n)$ is a mapping operator, which is a part of the definition of the CG model.

The CG potential $U(\mathrm{R}^N;\phi)$ is a linear combination of N_D basis functions of an appropriate type, where ϕ is a one-dimensional column vector of D parameters, ϕ_1, ..., ϕ_{N_D}, which are the coefficients of the basis functions. In the CG system, this potential determines the force on each site I

$$\mathbf{F}_I(\mathbf{R}^N;\phi) = -\frac{\partial}{\partial \mathbf{R}_I} U(\mathbf{R}^N;\phi) \tag{1}$$

Both $\mathbf{F}_I(\mathbf{R}^N;\phi)$ and $U(\mathbf{R}^N;\phi)$ are linear functions of the coefficients in ϕ. Obtaining a representation of those functions that can be used in a computer simulation of the CG system is the goal of the MS-CG calculation.

The MS-CG method provides the following prescription for determining the ϕ array and thereby obtaining the CG potential:

1. Perform molecular dynamics simulations of the atomistic system to obtain a large set of n_t configurations $\mathbf{r}^n$ that are representative of a canonical distribution of configurations for a specific temperature and volume. For each configuration, also obtain the force acting on each of the n atoms.

2. Calculate the locations of all the sites for each configuration.

3. Calculate a matrix $\mathbf{F}$ that has $3n_tN$ rows and N_D columns, each of whose elements is calculated from the positions of the N sites in one configuration and from one of the basis functions.

4. Calculate a column vector $\mathbf{f}$ that has $3n_tN$ elements, each of which is calculated from the forces acting on the atoms associated with one site in one atomistic configuration.

5. Determine

$$\arg\min_{\phi}(\mathbf{f} - \mathbf{F}\phi)^T(\mathbf{f} - \mathbf{F}\phi) \tag{2a}$$

which is the column vector ϕ that minimizes the function $(\mathbf{f} - \mathbf{F}\phi)^T(\mathbf{f} - \mathbf{F}\phi)$, which is a quadratic function of the entries of ϕ. Here, T denotes the transpose. This is equivalent to finding the ϕ array that approximately solves the overdetermined set of linear equations

$$\mathbf{F}\phi \cong \mathbf{f} \tag{2b}$$

in a least-squares sense. (The symbol $\cong$ is a reminder that the least-squares solution of the overdetermined set of equations is approximate, rather than exact.)

See refs 22 and 23 for further details.

2.2. Basis Functions. The choice of basis functions for the potential U and force functions $\mathbf{F}_I$ is made on the basis of physical intuition about the contributions that are expected to be important and to be easily representable. Many of them are expressed as functions of simple collective variables, such as the scalar distance between two sites. For example, if x is such a collective variable that is a function of $\mathbf{R}^N$, then a contribution to the potential $U(\mathbf{R}^N;\phi)$ might be of the form

$$\sum_d \phi_d f_d(x) \tag{3}$$

The corresponding contribution to the force $\mathbf{F}_I(\mathbf{R}^N)$ would then contain derivatives of the form $\mathrm{d}f_d(x)/\mathrm{d}x$. Depending on the problem, either the basis functions $f_d(x)$ or $f'_d(x)$, would be represented in some simple form.

In early MS-CG developments of Izvekov and Voth,[21] cubic spline basis functions were implemented for the force functions. Several other types of basis functions including linear spline and delta functions were also tested and compared in the work of Noid et al.[23] Generally speaking, these low-order spline functions are adequate for representing functions of one collective variable. Another set of basis functions was introduced by Das and Andersen to ensure that the force and its spatial derivatives are continuous.[31]

Low-order spline functions have the flexibility required for variational calculations of complicated CG interactions. However, typically, a large number of basis functions are needed for each type of interaction. Consequently, the number of unknowns in eq 2a can be very large, which adds to the computational challenges described in the first section. A natural solution for the problem is to incorporate higher-order polynomial basis functions and reduce the total number of basis functions needed. This has been achieved by applying B-spline functions as basis sets, introduced in the work of Lu and Voth.[29] It has been shown that both the computation time and required memory are reduced by using B-splines in MS-CG calculations, while at the same time

```
DO For all cells
  DO For all CG sites in the cell
    DO For other sites within the same cell
      IF Distance less than cutoff
        Compute forces and fill in matrix elements
      END IF
    ENDDO
    DO For sites in neighboring cells
      IF Distance less than cutoff
        Compute forces and fill in matrix elements
      END IF
    ENDDO
  ENDDO
ENDDO
```

Figure 1. Pseudocode for the neighbor search algorithm with the cell index method.

good accuracy can be retained.[29] Another advantage of B-spline basis functions is that the smoothness of the force function and some of its derivatives can be guaranteed, a desirable feature for filtering statistical noise in force data. However, in practice even with B-spline basis functions, the total number of unknowns in eq 2a can be very large, due to the complexity of realistic systems. Both linear spline and B-spline basis functions have therefore been implemented in the MSCGFM program to deal with different requirements for accuracy and efficiency.

3. Generating the Matrix Equation in Multiscale Coarse-Graining

3.1. Neighbor Search Algorithm. In order to perform the calculations, the vector **f** and the matrix **F** in eq 2a must be constructed. The vector **f** is easily calculated from the atomistic force data that is generated during the original atomistic simulation. The calculation of **F** requires identification of all sets of sites that are close enough to each other that one or more terms in $U(\mathbf{R}^N;\phi)$ can generate forces that act between them. Here, we will discuss only the algorithm for nonbonded interactions between pairs of sites that are close enough to interact.

When the system size is large, the direct search for each nonbonded pair of sites that are within a specific cutoff becomes very inefficient. For such a simple search algorithm, the number of interactions to be computed is $N(N-1)/2$, where N is the total number of CG sites in the system. Fortunately, the cell index method[32,33] widely used in MD simulations can also be implemented here. In this method, the simulation box is divided into a number of cells of a three-dimensional lattice. Each cell has a dimension that is equal to the cutoff distance. When the cell structure is applied, only $13.5NN_c$ interactions are calculated, where N_c is the average number of sites per cell. This is because the neighbor search is only performed for sites within the same cell and neighboring cells. In practice, the cell information is stored in a linked-list data structure, which enables a very efficient neighbor search with the cell index method. Once all the pairs are found, the matrix elements of **F** are calculated. The whole procedure is illustrated in Figure 1 as pseudocode. Although the approach in Figure 1 is specific for pairwise interactions, three-body CG interactions can be treated with a similar procedure. (Interactions that involve more than three sites are usually not practical in MD simulations.) In practice, the MS-CG calculation is dramatically slower without implementation of the cell index method. When the cell index method is used, the computation time for the generation of the **F** matrix step is much less than the time needed for performing the subsequent matrix calculations. Each atomistic configuration that is analyzed adds $3N$ rows to the matrix **F**. Depending on the method used to performed the matrix calculations (discussed later), the partial result for **F** from one configuration or a small block of configurations is converted to an intermediate matrix or a block solution for ϕ. The partial result for **F** is overwritten thereafter.

3.2. Dense and Sparse Matrix Formats. To perform the calculation associated with eq 2a, memory is required for both **F** and **f**, and some additional work space is needed. The memory requirement for storing **F** is by far the largest of the three. As will be shown in the next section, for most MD systems, **F** is a sparse matrix, allowing compact storage via sparse matrix formats. In the MSCGFM code, both dense (standard array data structure) and sparse formats are supported for different least-squares solvers. Although using the sparse matrix format usually saves space, the implementation of a dense matrix data structure makes it straightforward to use a large number of available packages for linear algebra calculations, such as LAPACK.[34] In addition, some equation solving algorithms require operations in dense matrix form, as will be discussed later in this article.

If **F** is very large and sparse, it is more appropriate to apply a sparse matrix format for storing and solving the minimization problem. This is the case for most complex multicomponent systems, especially biomolecular systems in which the number of interactions is very large. For some systems like proteins, the storage of the matrix in dense matrix form becomes impractical, which will be further demonstrated in the next section. It is natural to adopt the conventional Compressed Sparse Row (CSR) format[35] or similar formats since these formats are supported by most sparse linear algebra algorithms and software. In the CSR format, three arrays are needed for the number of nonzero elements in each row, the column indexes of the nonzero elements, and the nonzero element values, respectively. For both the column index array and element value array, both the length and the sequence of the array are fixed, and the memory occupation is continuous, as demonstrated in Figure 2a. These fixed properties of the arrays enable efficient matrix calculations since in these computations the memory access is continuous and the memory range is predefined. However, the CSR format is intrinsically not compatible with the neighbor search approach in Figure 1. In the neighbor searching method, the filling of the matrix elements is random and the total number of nonzero elements is unknown until the whole search procedure is finished. Because the array length is not predefined, allocation of memory then presents a challenge. Although in many programming languages such as C there are functions to dynamically allocate memory, this kind of dynamical allocation usually dramatically lowers computational efficiency in matrix computations. Besides the array length issue, the random filling of the matrix means that every element sequentially

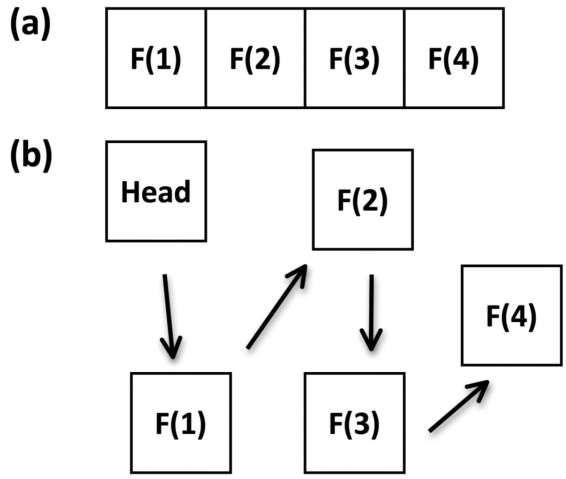

Figure 2. Memory allocations for (a) CSR format and (b) linked list format. In this example, there are four nonzero elements in the sparse matrix.

after the newly filled one must be moved in order to ensure the correct position of each element. This element moving process happens frequently during the neighbor search procedure and can make the whole neighbor search algorithm impractical. Therefore, another sparse matrix format, namely, the linked list format,[36] is implemented for the matrix **F** during the neighbor search. The discontinuous memory occupation of the linked list format is demonstrated in Figure 2b. As shown in that figure, there are no arrays needed in the linked list format, and the matrix elements are stored as scattered "nodes" in memory. For each node, both the element information and the address for the next node are stored. The address information ensures that each node in the linked list can be found sequentially. In addition, there is a "head" node that serves as the starting point of the linked list and stores information like the total number of nonzero elements. Two advantages exist for this linked list data structure in the neighbor search. First, the length of the linked list does not need to be predefined, and it is determined by the address information of the last node. Memory space for each node is created when the new node is generated in the linked list format, which allows efficient usage of computer memory. Second, random insertion of matrix elements becomes computationally efficient since only the address information of the previous node needs to be modified during the insertion. Once the matrix **F** is created after the neighbor search and stored in the linked list format, it can be converted to the conventional CSR format and then is ready to be processed by many sparse linear algebra algorithms. In this equation generation approach with the linked list format, two copies of matrix **F** are created in both linked list and CSR formats. The gain in computation efficiency far outweighs the memory cost of maintaining two copies.

4. Solving the Least-Squares Problem in Multiscale Coarse-Graining

There are a number of established ways to perform the minimization in eq 2a.[37] Due to some special numerical properties of the **F** matrix, several different strategies are discussed here.

4.1. Properties of the Least-Squares Problem. *Matrix Dimension.* Table 1 shows row/column dimensions of the matrix **F** for several simple liquid and biomolecular systems. It is seen that, for simple liquids like water, **F** is rather small, which indicates that eq 2a can be solved rapidly. However, the size of the matrix increases dramatically with system complexity. All biomolecular systems in Table 1 require memory on the order of gigabytes to store the matrix **F**. Note that for the numbers in Table 1 only 10 configurations of the all-atom trajectory are counted. In practice, at least hundreds to tens of thousands of configurations are needed to ensure good statistical force sampling, depending on the complexity of the system. Since the size of **F** is proportional to the number of configurations, in real MS-CG calculations the memory requirement for **F** is usually beyond the hardware limit, even considering sparsity.

Sparsity. Table 1 also shows the percentage of nonzero elements in the matrix **F** for different systems. Traditionally, a matrix is considered to be sparse when the fraction of nonzero elements is less than 5%. For simple liquid systems like water, the matrix is typically not sparse. By contrast, matrices for complicated systems are usually very sparse. This phenomenon is understandable since the number of nonzero interactions is limited for each CG site due to the short range of most of the interactions. For complex systems, the total number of force functions, which corresponds to the column dimension of **F**, is a very large number. Each basis function is typically nonzero only for a small range of its arguments. Therefore, many elements of the **F** matrix are zero. This sparse nature of **F** points to the need to implement sparse linear algebra algorithms, to be discussed later.

Ill-Posed Problems. For many inverse problems, the related least-squares problems are ill-posed. Typically, there are two categories of ill-posed problems, called rank-deficient and discrete ill-posed problems, which are distinguished from one another on the basis of singular value analysis.[38]

In a singular value decomposition (SVD),[39] the matrix **F** can be expressed as product of three matrices:

$$\mathbf{F} = \mathbf{USV}^T \tag{4}$$

where **U** and **V** are orthogonal matrices and **S** is a diagonal matrix with diagonal elements that are called the singular values of the matrix **F**. Conventionally, the singular values S_i are ordered in a nonincreasing fashion, and the condition number is defined as S_{max}/S_{min}. For a rank-deficient problem, there exists a cluster of small singular values, and the gap between small and large singular values is obvious. In this case, the small singular values are usually set to zero in order to regularize the least-squares solution, which is the so-called truncated SVD method.[40]

In the discrete ill-posed problem case, the singular values decrease gradually without any obvious gap. Consequently, various regularization methods need to be implemented in order to balance the residual norm and solution size. Note that common regularization methods used for discrete ill-posed problems can also be applied to rank-deficient cases.

The singular values of a water system and a lipid bilayer system are plotted in Figure 3. From Figure 3, it is seen that the MS-CG problem for the water system is rank-deficient

Table 1. Matrix Dimension and Non-Zero Element Ratio for the MS-CG Calculations for Selected Molecular Systems[a]

system	CG type	n_{row}	n_{col}	$n_{row} \times n_{col}$	memory	nonzero element ratio (%)
1000 water	1	30 000	161	4 830 000	4.6 MB	23
128 lipid (DOPC/DPPC 1:1) + 4000 water	15	173 760	35 031	6 086 986 560	5.7 GB	0.24
A polypeptide + 6256 water	19	199 530	35 058	6 995 122 740	6.5 GB	0.17
T4 Lysozyme + 9739 water	22	297 150	49 292	14 647 117 800	13.6 GB	0.13

[a] For each system, 10 frames of all-atom trajectory are included in the calculation.

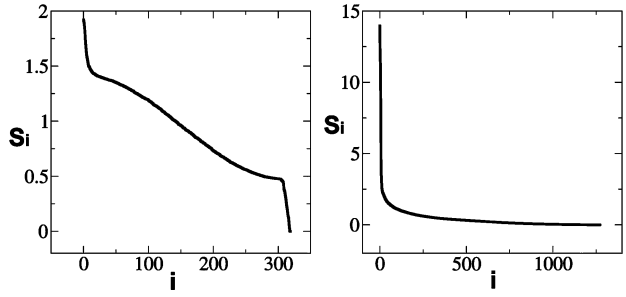

Figure 3. Singular value distributions for (left) a 1000 water system and (right) a 64 DPPC + 3846 water system.

since the smallest singular value 2.9×10^{-16} is by far smaller than any of the other singular values. It is also seen from Figure 3 that the one for the lipid bilayer is a discrete ill-posed problem.

The singular value analysis results can guide one to choose an appropriate least-squares solver and regularization method. Noid et al.[23] point out that, by removing basis functions that are related to unsampled interaction distances, the condition number of **F** can be reduced. Appropriate column rescaling may also reduce the condition number.[23] However, due to statistical noise and redundant information from all-atom configurations,[41] the MS-CG least-squares problem is still typically ill-posed after applying such treatments. In practice, for systems like simple liquids, the MS-CG least-squares problem is usually with full-rank or rank-deficient. For complex systems, rank-deficient and discrete ill-posed problems are often encountered.

4.2. Least Squares Algorithms. From the previous discussion about the matrix size of **F**, it is clear that it is impractical to store the whole **F** in memory, even with sparse matrix formats. Because a large number of all-atom configurations are usually needed, even various out-of-core least-squares algorithms are not able to handle the resultant huge matrix due to hard disk limitations. As a result, two categories of least-squares algorithms have been developed for solving the least-squares problem. The first type of method is based on the "block average" approximation, which is the one used by Izvekov and Voth in the early MS-CG applications.[21] In the second type, the huge size matrix eq 2a is first converted to a smaller equation that is numerically equivalent to eq 2a. Then, the smaller equation is solved by various standard methods. For this category, there are two major algorithms called the normal equation method[39] and the sequential accumulation method,[37] which will be discussed later.

Block Average Algorithm. The basic idea of the block average algorithm is to divide the all-atom trajectory into small blocks of configurations, minimize eq 2a for each block, and obtain the ϕ vector for each block. The last step of the block average algorithm is to calculate the average of the ϕ vectors for the individual block as the final approximate MS-CG solution. Noid et al.[23] pointed out that, if there are significant structural transitions that take place on a time scale that is longer than the block size, there might be systematic error from the block average approach. More theoretical investigation is therefore needed on the applicability of the block average method. However, numerical results from the MS-CG applications to date show that the method is a reasonable approximation for many systems and a discussion can be found in the literature.[23] The block average method is also the only method in which it is possible to use sparse solvers since in this method small problems in the form of eq 2a are directly solved and the sparsity of **F** can be utilized by least-squares solvers. The other algorithms involve matrix transformations that may destroy the sparsity of the relevant matrices.

Normal Equation Algorithm. It is straightforward to show that any solution ϕ of the minimization problem in eq 2a is also a solution of the following equation:

$$\mathbf{F}^T\mathbf{F}\phi = \mathbf{F}^T\mathbf{f} \tag{5}$$

Moreover, the solution of eq 5 is unique if and only if the solution of eq 2a is unique. $\mathbf{F}^T\mathbf{F}$ is an $N_D \times N_D$ matrix (much smaller than **F**, which is $3nN \times N_D$), and so algorithms based on solving eq 5 require less memory than those that deal with eq 2a. The matrix $\mathbf{F}^T\mathbf{F}$ can be constructed directly from the atomistic configuration data without first constructing **F**.

Equation 5 is a set of linear equations that in principle can be solved exactly. Direct methods of solution as well as least-squares methods can be applied. Since the least-squares solver needs to be applied only once to solve eq 5, for small to medium sized systems (i.e., those with up to around 10 types of CG sites), the normal equation algorithm is faster than the block average method. Unlike the case of the block average algorithm, the normal equation algorithm in principle gives the exact solution of the variational problem in eq 2a. However, in matrix computations, the machine error and statistical noise from data need to be considered. The condition number of $\mathbf{F}^T\mathbf{F}$ is the square of the condition number of **F**. Therefore, it is usually not recommended to work with eq 5 for the high condition number problems[37,38] that are often seen in MS-CG applications. Another disadvantage of the normal equation algorithm is that the information of the residual of eq 2a is lost during the normal equation transformation. Although this information is not necessary for a CG force field, it is an important quantity when a regularization method needs to be implemented. The details of regularization will be discussed later in this article.

Sequential Accumulation Algorithm. In the sequential accumulation algorithm introduced by Lawson and Hanson,[37] the precision and condition number issues in the normal

equation algorithm do not exist. The key idea of this algorithm is to convert eq 2a to a smaller least-squares problem as[37]

$$\begin{bmatrix}\mathbf{R}\\0\end{bmatrix}\phi \cong \begin{bmatrix}\mathbf{d}\\e\end{bmatrix} \tag{6}$$

where $\mathbf{R}$ is a upper triangular matrix with dimension $N_D \times N_D$, $\mathbf{d}$ is a vector of N_D length, and e is a real number. This transformation is achieved through a number of QR decompositions in which $\mathbf{F}$ needs to be divided into blocks. Besides obtaining the same set of solutions, eq 6 has some desirable numerical properties. First, it has the same residual as that in the original least-squares problem eq 2a, and the residual is simply the value of e in eq 6. Second, the matrix $\mathbf{R}$ has the same set of singular values as $\mathbf{F}$, and there is no precision loss as in the normal equation algorithm. Due to the above properties, the sequential accumulation algorithm is usually the first choice for very ill-posed problems since the condition number will not increase during the transformation from eq 2a to 6, and various regularization methods can be easily implemented. The memory requirement and computational efficiency of this algorithm highly depend on the block size during the transformation. Generally speaking, the smaller the block size, the smaller the memory space required and the more computation time needed. During the transformation, memory is required for both $\mathbf{R}$ and the block of $\mathbf{F}$. Typically, the sequential accumulation algorithm is slower than the normal equation algorithm.

4.3. Least Squares Solvers. There are a number of standard solvers for solving the least-squares problems, and the choice of solvers largely depends on the availability of optimized software. Many dense solvers such as Gaussian elimination are not robust for problems whose matrix does not have full rank. Thus, they cannot be chosen to solve the MS-CG problem. Two common dense solvers for rank-deficient problems are SVD and QR decomposition with pivoting.[39] For dense matrix algorithms like the normal equation algorithm and sequential accumulation algorithm, the solver is used only once during the whole MS-CG calculation, and the speed of the solver is not very important. Therefore, the SVD method is implemented in this case, although it is slower than the method of QR decomposition with pivoting. On one hand, SVD is a good choice for high condition number problems that are often encountered in MS-CG applications. On the other hand, the results of singular value analysis identify problems of an ill-posed nature. Sparse solvers can be used with the block average algorithm in MS-CG calculations. In this case, the least-squares QR (LSQR) algorithm[42] is usually implemented since it is known to be a robust solver for ill-posed problems.

Various algorithm/solver strategies in the MSCGFM program are listed in Figure 4. For very complicated systems such as proteins with around 20 CG site types, the block average algorithm with sparse matrix format and LSQR solver is usually the only computationally affordable choice due to the very large memory requirement. The sequential accumulation algorithm is good at very ill-posed problems. When the matrix $\mathbf{F}$ is not very large and ill-conditioned, the normal equation algorithm is often a good choice since it is usually faster than the sequential accumulation algorithm. Both the normal equation and sequential accumulation algorithms involve intermediate dense matrices. Therefore, only dense solvers can be implemented in these cases, and the SVD solver is usually chosen.

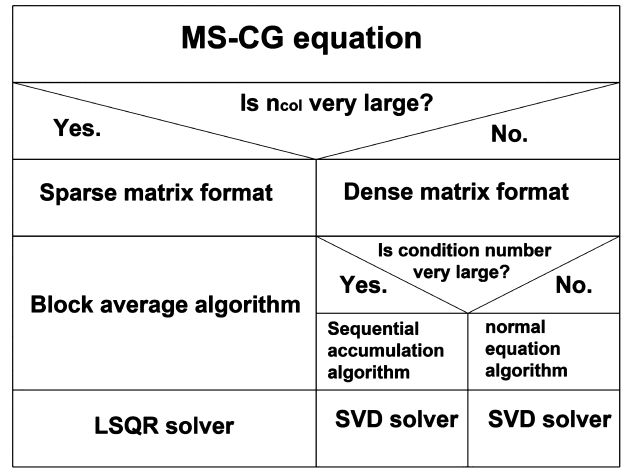

Figure 4. Various choices of MS-CG data structures, algorithms, and least-squares solvers.

4.4. Parallelization. If an efficient neighbor search algorithm is used in the construction of the $\mathbf{F}$ matrix, the most time-consuming part of the MS-CG calculation is the solution of the least-squares problem in eq 2a. Thus, parallel algorithms are applied only to the equation solving process. Since there are different equation solving algorithms for different types of MS-CG applications, separate parallelization strategies have to be developed for each case.

In the block average algorithm, the computation for each block is intrinsically independent. This makes the parallelization strategy quite straightforward. To parallelize the MS-CG calculation in this case, a number of independent processes are created, and each single process is responsible for the computation of a certain number of blocks. The average block solution from each process is then stored on hard disk, and the final average can be calculated from these solutions. No communication is necessary between any of the processes, so the parallel efficiency can be considered to be 100%. One disadvantage of this parallel strategy is that each process needs the same amount of memory as that for the serial computation. However, the block average algorithm is usually used for very large systems, and from Table 1 it can be seen that the matrix $\mathbf{F}$ is always very sparse in this case. As a result, the MS-CG calculation with sparse matrix format is not extremely demanding for memory, and the above strategy works in most applications. In principle, MPI-based sparse solvers may be implemented for problems with extremely large memory requirements that cannot be handled by the above strategy. However, the upper bound for the number of CG site types is around 20 to 30 for protein systems studied to date, and in practice these applications are tractable by the above parallelization strategy. In addition, the MPI-based sparse solver may reduce the parallel efficiency due to communication requirements. In the MSCGFM program, the parallelization for sparse algorithms is realized by the above strategy of independent processes, while the MPI-based algorithm is still under development.

The normal equation algorithm is parallelized by the same approach with independent processes. The MS-CG computation is divided into a number of processes for blocks of all-atom trajectory frames. The intermediate information that must be saved is block results for $\mathbf{F}^T\mathbf{F}$ and $\mathbf{F}^T\mathbf{f}$. The final results of $\mathbf{F}^T\mathbf{F}$ and $\mathbf{F}^T\mathbf{f}$ are then calculated on the basis of the block results from individual processes. Compared to the block average algorithm, the required hard disk space for intermediate results is much larger since both the left- and right-hand side information must be stored. This hard disk requirement is usually not a serious problem because the normal equation algorithm is used mostly for small systems.

In the sequential accumulation algorithm, the all-atom trajectory must be treated sequentially. Hence, it is not possible to divide the MS-CG calculation into independent processes and apply the above parallelization strategy. However, since the dense QR decomposition involved in the algorithm is included in MPI-based linear algebra software, such as SCALAPACK,[43] it is possible to implement MPI-based solvers in this case. Because of the need for interprocess communication, the parallel efficiency in this case may be much less than 100%, depending on the matrix structure and hardware/software configurations. However, since the sequential algorithm is usually used for small systems, in most applications no internode parallelization is necessary.

So far, the discussed parallelization strategies can be applied for both distributed and shared memory hardware. Presently, computer clusters with multi-CPU/multicore nodes and high speed internode connections are the standard in high performance scientific computing. Multicore CPUs have also become popular in workstations. In these cases, the computing node or CPU can be considered a computing unit with shared memory. Thus, it is desirable to implement parallelization for shared memory systems that benefits from the hardware structure. These parallelization approaches, such as the widely used OpenMP library,[44] can be used with the distributed memory parallelization discussed above or used alone for small- to medium-size problems. Fortunately, the Intel Math Kernel Library (MKL)[45] supports threaded LAPACK functions for linear algebra computations through the OpenMP environment. This is utilized for dense matrix algorithms in MS-CG calculations, and both the normal equation algorithm and the sequential accumulation algorithm are threaded.

4.5. Regularization. Since eq 2a is ill-possed in most medium- to large-size MS-CG applications, it is necessary to consider numerical regularization methods. For rank-deficient problems, simple regularization methods like the truncated SVD method that is implemented with the SVD solver in MS-CG calculations can generate reasonable results. However, more sophisticated regularization must be implemented for discrete ill-posed problems in MS-CG applications. For example, Liu et al.[46] successfully implemented a Bayesian regularization approach for the MS-CG problem. The results from Liu et al. show that the CG force curve in badly sampled interaction distances can be dramatically improved through the Bayesian treatment. However, the regularization method introduced here is different compared to that from Liu et al. in the following ways: First, the regularization from Liu et al. is designed for square equations; i.e., it is designed for the normal eq 5, whose matrix has a larger condition number than the matrix in eq 2a. The residual information that is important for regularization is also lost during the conversion to the normal equation. Second, the algorithm of Liu et al. involves a matrix inversion of $\mathbf{F}^T\mathbf{F}$. This makes the computation very expensive for large systems since the matrix inversion calculation is well-known to be slow.[47] Finally, the iterative scheme proposed by Liu et al. might converge slowly for complicated systems. The goal here is therefore to apply a regularization scheme that acts on the original eq 2a and is computationally inexpensive.

The Tikhonov regularization with L-curve criteria[38] for choosing regularization parameters is very successful for many ill-posed problems and is therefore appropriate to regularize eq 2a. In Tikhonov regularization, instead of solving eq 2a, one solves a regularized equation

$$\begin{bmatrix}\mathbf{F}\\ \lambda\mathbf{I}\end{bmatrix} \cong \begin{bmatrix}\mathbf{f}\\ 0\end{bmatrix} \tag{7}$$

Here, λ is a regularization parameter and $\mathbf{I}$ is the identity matrix. By adding the regularization term $\lambda\mathbf{I}$, the solution becomes smoother and the residual norm becomes larger. For the original eq 2a, a direct solution is often dominated by contributions from noise and round-off errors, and adding the regularization term can solve this problem at the cost of a slightly increased residual. Hence, the main objective in Tikhonov regularization is to choose a reasonable parameter λ in order to obtain the balance between solution accuracy and smoothness. This can be achieved by applying the L-curve criteria in which λ is chosen to give the "corner" of the residual norm-solution norm curve on a logarithm scale plot, where the corner is defined as the point with maximum curvature. In a standard L-curve procedure, a number of values of λ are chosen, and the corresponding residual norms and solutions norms are calculated. The results are then plotted on a log−log scale, and the corner point is found after numerical interpolation of the curve using splines. There are a few drawbacks, however, for applying this procedure in MS-CG calculations. First, the matrix $\mathbf{F}$ is not stored during the MS-CG calculation, and for each λ, it needs to be rebuilt by scanning the whole all-atom trajectory. Second, eq 7 must be solved for a number of different values of λ, which is the most expensive part of the L-curve calculation. Finally, the result of the corner point strongly depends on the interpolation scheme. Because calculating one point on the L-curve is expensive, usually only a few points on the plot can be obtained, and the interpolation accuracy can be difficult to guarantee.

Fortunately, if SVD can be performed for $\mathbf{F}$, the curvature for a point on the L-curve can be expressed analytically as a function of λ and singular values and vectors.[48,49] In this way, the procedure to find the corner turns out to be a one-dimensional minimization problem involving λ. Therefore, the atomistic trajectory needs to be read only once, and only one SVD needs to be performed for the original matrix $\mathbf{F}$. This seems to be much more efficient than the above approach of computing each L-curve point. However, the

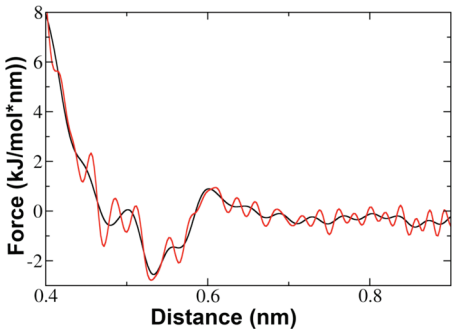

Figure 5. The CG forces for water−water interactions. The red line is the original MS-CG force, and the black line is the force after regularization. Only force values in the range of 0.4 to 0.9 nm are plotted to clearly show the overfitting phenomenon.

huge size of **F** makes the direct SVD prohibitive in most cases. Recall that in the sequential accumulation algorithm the matrix **R** in eq 6 has the same singular values as **F**, and eq 6 has the same solution and residual as the original eq 2a. Thus, regularization can be applied to eq 6 instead as

$$\begin{bmatrix} \mathbf{R} \\ 0 \\ \lambda\mathbf{I} \end{bmatrix} \cong \begin{bmatrix} \mathbf{d} \\ e \\ 0 \end{bmatrix} \tag{8}$$

in which SVD of **R** is usually affordable.[37] For extremely large systems that are usually handled by the block average algorithm with a sparse solver, it is still possible to convert the L-curve corner-finding procedure to a one-dimensional minimization problem involving λ. However, a least-squares equation with a size similar to eq 2a needs to be solved for each λ, and the whole λ optimization case is much more expensive than the approach with SVD of eq 6.[48]

A numerical example of a system with 1000 TIP3P[50] water molecules is shown in Figure 5. In this example, a B-spline basis function with the order $k = 40$ and spacing 0.01 nm is applied. An unreasonably high order B-spline is used to demonstrate the possibility of the overfitting phenomenon in MS-CG calculations. Since the matrix for this application is small, it is possible to do SVD for the left-hand side matrix in eq 6. Therefore, the curvature of the L-curve can be calculated analytically, and the resultant one-dimensional optimization value for λ is 1.5849. The black line in Figure 5 is the CG force curve after regularization with the λ value. It is shown in Figure 5 that the original CG force curve has many unphysical fluctuations corresponding to overfitting. However, the force curve is smoothed after regularization, and these artificial fluctuations disappear. Since the statistical sampling for this CG water system is reasonable, it is hard to observe the effect of regularization until a very large k is used, and no regularization is needed for a normal k value. It is expected that for other more ill-posed MS-CG problems the difference between the results with and without regularization appears to be more obvious, and regularization becomes necessary.

5. Miscellaneous Algorithms

In MS-CG calculations, most basis functions in eq 3 are spline functions. In order to define a spline interpolation, the minimum and maximum distances for each interaction need to be determined. A complete search for these distance ranges is therefore performed before defining spline basis sets. There are two advantages for doing the distance search rather than guessing the ranges arbitrarily. On one hand, this search assures minimal memory spaces are allocated for basis functions. Spline basis functions are assigned to sampled distances only after the search. On the other hand, excluding basis functions corresponding to unsampled distances reduces the condition number of the matrix **F** as pointed out by Noid et al.[23] For large systems, the distance searching can be time-consuming, though it is still much faster than solving the least-squares equation. Since only minimum and maximum distances for each interaction are needed, the distance searching can be divided into independent processes for small trajectory blocks, which is straightforward to be parallelized for large systems. Sometimes RDFs or bond/angle/dihedral distributions are calculated for other purposes. In this case, distance ranges can be extracted from distribution function data, and the distance search is no longer necessary.

In the MS-CG application to ionic liquids by Wang et al.,[24] the bonded interactions are determined from an inverse Boltzmann approach because the bonded parameters are usually more affected by statistical noise. CG parameters not easily obtained from the MS-CG scheme can typically be determined by inverting distribution functions or using some knowledge based approach. The iterative procedure suggested by Wang et al.[24] can be applied to determine those CG parameters. The MSCGFM code allows an arbitrary set of the ϕ coefficients to be chosen in advance, input into the calculation, and held fixed in the variational calculation. This makes it possible to have the MS-CG variational principle determine only a few critical CG interactions for complex systems, and the computation expense can be reduced. The variational calculation with some coefficients held fixed is of the same form as eq 2a, with a new **F** matrix with fewer columns than the original matrix, a new ϕ matrix that includes the parameters that are to be varied, and a new **f** vector with the same length as the original vector.

One advantage of specifying some interaction parameters in advance and keeping them fixed in the variational calculation is that it speeds up the MS-CG calculation, especially when a sparse solver is used. In most biomolecular all-atom systems, the number of solvent molecules is very large, and in the corresponding CG system most CG sites are solvent sites. Consequently, most elements in **F** are determined by forces acting on the solvent in the atomistic simulations. If the solvent−solvent interaction parameters are not included in the variational calculation, the new **F** matrix has many fewer nonzero elements than the original **F**, and the MS-CG calculation is much less expensive for a sparse solver. Usually the solvent−solvent interaction is obtained from a separate MS-CG calculation for a pure solvent system. For dense solver cases, removing solvent−solvent interactions from the variational calculation does not change the computation time dramatically since the matrix dimension is only changed slightly.

Table 2. MS-CG Calculation Benchmarks for Selected Molecular Systems[a]

system	basis function	CG type	speed (frames/hour)
Water	linear spline	1	4.99×10^4
DPPC	linear spline	7	4.32×10^2
DPPC	B-spline	7	3.26×10^3
T4 lysozyme	B-spline	32	19.5

[a] Details of each system are described in the main text.

6. Benchmarks

Three systems were chosen to generate benchmarks of the MSCGFM program. The first system contained 1000 TIP3P water molecules,[50] and a CG mapping was applied to this system to obtain a one-site CG water model. For this system, a linear spline of a grid spacing of 0.005 nm was applied. The second system contained 64 DPPC lipids and 3846 TIP3P water molecules.[50] A solvent-free DPPC lipid CG model was then generated from the all-atom systems, and the lipid CG mapping was similar to that used by Izvekov and Voth.[26] As a result, there were 960 CG sites and 7 CG site types in the CG system. Bonds and angles were treated as 1−2 and 1−3 two-body interactions, respectively, and the CG potential contains no explicit dihedral terms.[26] Two different types of basis sets were applied to this system. The first type were linear spline basis functions with a spacing of 0.01 nm for the nonbonded interactions and 0.005 nm for the bonded interactions. The second type were B-spline basis functions with a spacing of 0.04/0.01 nm and B-spline order k as 6/4 for the nonbonded/bonded interactions. In practice, these B-spline parameters give a CG force field result that is very similar to that from linear splines. The normal equation algorithm and SVD dense solver were applied for the above two systems to solve the least-squares problem. The third system contained a bacteriophage T4 lysozyme and 9733 explicit water molecules. After the all-atom to CG mapping there were 9771 CG sites and 32 CG site types in the CG system. All adjacent protein CG sites were connected by bonds, which made the total number of bonds 29. The B-spline basis functions for this system had a spacing of 0.03 nm for the nonbonded interactions and 0.01 nm for the bonded interactions. The B-spline order was 4 for all interactions. This application was to show the program's ability to treat very large systems with the water−water interaction inputted from a table for computational efficiency. The MS-CG calculation for this protein system was performed with the block average algorithm and LSQR sparse solver.

Table 2 shows the computation speed for each MS-CG calculation for the three tested systems. All computations in Table 2 were performed on a workstation with Intel dual core Xeon 2.0 GHz CPUs. Only one core of the CPUs was used for each calculation, and the Intel MKL package[45] was used for dense matrix computations. The LSQR computation was performed using a compiled FORTRAN 77 code. Computational speed is defined by the number of all-atom configurations (frames) that are processed per hour. It is seen in Table 2 that MS-CG calculations for systems with less than 10 CG site types are usually very efficient on a single processor. Note that for most MS-CG calculations at least several thousand all-atom configurations are necessary. For better statistical sampling and smoother output force curves, tens of thousands of atomistic configurations are desirable depending on the system complexity. On the basis of the speed data in Table 2, these requirements can be easily achieved for small systems. However, the matrix column dimension N_D is proportional to the square of the number of CG site types, which causes the required computer time to increase dramatically when the number of CG site types becomes large. For example, the third benchmark system has 32 CG site types, and it is extremely hard to solve this MS-CG problem using dense matrix algorithms considering the memory and computer time required. For this large system, the required computer time even prohibits calculation with a sparse algorithm. Therefore, the water−water interaction is inputted from an external table, which dramatically speeds up the calculation. The speed results show it is able to perform the MS-CG calculation for this complex system using tens of CPUs and a few days. Thus, this application is a good example to demonstrate MSCGFM's ability to treat complex systems.

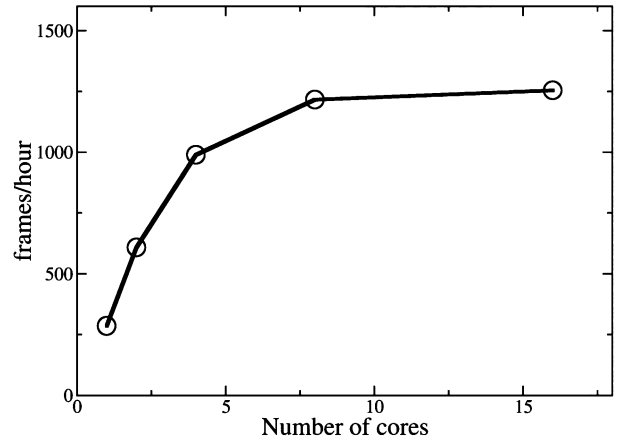

Figure 6. Scaling of an OpenMP parallelized MS-CG calculation for a DPPC lipid CG system.

From Table 2, it is also seen that for small- to medium-sized MS-CG systems with around 10 CG site types, applying B-spline basis functions can greatly increase the computation efficiency. This is because the matrix column dimension is reduced by using a smaller number of basis functions. For very large problems with a sparse algorithm, the computation speed depends on both matrix dimension and nonzero element percentage. Using higher-order basis functions usually makes the matrix **F** less sparse. Therefore, there is no straightforward relation between the type of basis functions and the computation efficiency if a sparse algorithm is used.

Figure 6 shows the scaling for dense normal equation algorithms with OpenMP support. The system is the DPPC system described earlier with linear spline basis functions. These calculations were performed on 16-core computing nodes on an AMD Opteron cluster with OpenMP supported Intel MKL software. It is clearly seen that the calculation scales well up until 8 cores. Considering that computing nodes on most high performance computing clusters are comprised of 2 to 16 cores, dense matrix MS-CG calculations can be well parallelized for these hardware structures.

7. Conclusions

A number of algorithms for MS-CG calculations have been introduced in the recently developed MSCGFM program. The MS-CG approach has been demonstrated to be a successful method to generate CG force field parameters from all-atom trajectories. However, computational efficiency and stability are critical issues when applying the MS-CG method to complex systems. In order to speed up the calculation, various dense and sparse matrix algorithms, which are appropriate for different types of CG systems, are incorporated in the MSCGFM code. In particular, the implementation of sparse algorithms enables the MS-CG program to deal with complicated biomolecular systems, such as proteins. Tikhonov regularization is also proposed to regularize ill-posed MS-CG problems for certain CG systems. Figure 4 shows a flowchart for the implementation of the code for a given MS-CG calculation.

Future work will involve the development of better parallelization approaches for the MS-CG calculations on distributed memory machines, especially for the sparse matrix algorithms. New basis functions corresponding to various physical interactions will also be incorporated.

The code of MSCGFM is available on request and will soon be released under a public license.

Acknowledgment. This research was supported by a Collaborative Research in Chemistry grant from the National Science Foundation (CHE-0628257). Computer resources were provided by the National Science Foundation through TeraGrid computing resources administered by the Pittsburgh Supercomputing Center, the San Diego Supercomputer Center, the National Center for Supercomputing Applications, the Texas Advanced Computing Center, and Argonne National Laboratories. The authors gratefully acknowledge Dr. Edward Lyman, Dr. Ron Hills and Dr. Sven Jakobtorweihen for critical reading of the manuscript.

References

(1) Brooks, B. R.; Bruccoleri, R. E.; Olafson, D. J.; States, D. J.; Swaminathan, S.; Karplus, M. *J. Comput. Chem.* **1983**, *4*, 187.

(2) MacKerel, Jr., A. D.; Brooks III, C. L.; Nilsson, L.; Roux, B.; Won, Y.; Karplus, M. In *CHARMM: The Energy Function and Its Parameterization with an Overview of the Program*; John Wiley & Sons: Chichester, 1998; Vol. 1; pp 271.

(3) Case, D. A.; Cheatham, T. E., III; Darden, T.; Gohlke, H.; Luo, R.; Merz, K. M., Jr.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J. *J. Comput. Chem.* **2005**, *26*, 1668.

(4) Jorgensen, W. L.; Tirado-Rives, J. *J. Am. Chem. Soc.* **1988**, *110*, 1657.

(5) Schuler, L. D.; Daura, X.; van Gunsteren, W. F. *J. Comput. Chem.* **2001**, *22*, 1205.

(6) Tozzini, V. *Curr. Opin. Struct. Biol.* **2005**, *15*, 144.

(7) Ayton, G. S.; Noid, W. G.; Voth, G. A. *Curr. Opin. Struct. Biol.* **2007**, *17*, 192.

(8) *Coarse-Graining of Condensed Phase and Biomolecular Systems*; Voth, G. A., Ed.; CRC Press: Boca Raton, FL, 2008.

(9) Marrink, S. J.; Risselada, H. J.; Yefimov, S.; Tieleman, D. P.; de Vries, A. H. *J. Phys. Chem. B* **2007**, *111*, 7812.

(10) Monticelli, L.; Kandasamy, S. K.; Periole, X.; Larson, R. G.; Tieleman, D. P.; Marrink, S.-J. *J. Chem. Theory Comput.* **2008**, *4*, 819.

(11) Lyubartsev, A. P.; Laaksonen, A. *Phys. Rev. E* **1995**, *52*, 3730.

(12) Reith, D.; Putz, M.; Muller-Plathe, F. *J. Comput. Chem.* **2003**, *24*, 1624.

(13) Murtola, T.; Bunker, A.; Vattulainen, I.; Deserno, M.; Karttunen, M. *Phys. Chem. Chem. Phys.* **2009**, *11*, 1869.

(14) Henderson, R. L. *Phys. Lett. A* **1974**, *49*, 197.

(15) Chayes, J. T.; Chayes, L.; Lieb, E. H. *Commum. Math. Phys.* **1984**, *93*, 57.

(16) Chayes, J. T.; Chayes, L. *J. Stat. Phys.* **1984**, *36*, 471.

(17) Lyubartsev, A. P.; Laaksonen, A. *Phys. Rev. E* **1997**, *55*, 5689.

(18) Shell, M. S. *J. Chem. Phys.* **2008**, *129*, 144108.

(19) Johnson, M. E.; Head-Gordon, T.; Louis, A. A. *J. Chem. Phys.* **2007**, *126*, 144509.

(20) Izvekov, S.; Voth, G. A. *J. Phys. Chem. B* **2005**, *109*, 2469.

(21) Izvekov, S.; Voth, G. A. *J. Chem. Phys.* **2005**, *123*, 134105.

(22) Noid, W. G.; Chu, J.-W.; Ayton, G. S.; Krishna, V.; Izvekov, S.; Voth, G. A.; Das, A.; Andersen, H. C. *J. Chem. Phys.* **2008**, *128*, 244114.

(23) Noid, W. G.; Liu, P.; Wang, Y.; Chu, J.-W.; Ayton, G. S.; Izvekov, S.; Andersen, H. C.; Voth, G. A. *J. Chem. Phys.* **2008**, *128*, 244115.

(24) Wang, Y. T.; Izvekov, S.; Yan, T. Y.; Voth, G. A. *J. Phys. Chem. B* **2006**, *110*, 3564.

(25) Liu, P.; Izvekov, S.; Voth, G. A. *J. Phys. Chem. B* **2007**, *111*, 11566.

(26) Izvekov, S.; Voth, G. A. *J. Chem. Theory Comput.* **2006**, *2*, 637.

(27) Zhou, J.; Thorpe, I. F.; Izvekov, S.; Voth, G. A. *Biophys. J.* **2007**, *92*, 4289.

(28) Thorpe, I. F.; Zhou, J.; Voth, G. A. *J. Phys. Chem. B* **2008**, *112*, 13079.

(29) Lu, L. Y.; Voth, G. A. *J. Phys. Chem. B* **2009**, *113*, 1501.

(30) Izvekov, S.; Voth, G. A. *J. Phys. Chem. B* **2009**, *113*, 4443.

(31) Das, A.; Andersen, H. C. *J. Chem. Phys.* **2009**, *131*, 034102.

(32) Hockney, R. W.; Eastwood, J. W. *Computer simulation using particles*; Taylor & Francis, Inc.: Bristol, PA, USA 1988.

(33) Quentrec, B.; Brot, C. *J. Comput. Phys.* **1975**, *13*, 430.

(34) Anderson, E.; Bai, Z.; Bischof, C.; Blackford, L. S.; Demmel, J.; Dongarra, J. J.; Croz, J. D.; Hammarling, S.; Greenbaum, A.; McKenney, A.; Sorensen, D. *LAPACK Users' guide (third ed.)*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1999.

(35) Saad, Y. *Iterative Methods for Sparse Linear Systems*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2003.

(36) Antonakos, J. L.; Mansfield, K. C. *Practical Data Structures Using C/C++ with 3.5 Disk*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 1998.

(37) Lawson, C. L.; Hanson, R. J. *Solving Least Squares Problems*; Prentice-Hall, Englewood Cliffs, NJ, 1974.

(38) Hansen, P. C. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1998.

(39) Golub, G. H.; Van Loan, C. F. *Matrix computations* (3rd ed.); Johns Hopkins University Press: Baltimore, MD, USA 1996.

(40) Hansen, P. C. *BIT* **1987**, *27*, 534.

(41) Savelyev, A.; Papoian, G. A. *J. Phys. Chem. B* **2009**, *113*, 7785.

(42) Paige, C. C.; Saunders, M. A. *ACM Trans. Math. Softw.* **1982**, *8*, 195.

(43) Blackford, L. S.; Choi, J.; Cleary, A.; D'Azeuedo, E.; Demmel, J.; Dhillon, I.; Hammarling, S.; Henry, G.; Petitet, A.; Stanley, K.; Walker, D.; Whaley, R. C. *ScaLAPACK user's guide*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1997.

(44) Chandra, R.; Dagum, L.; Kohr, D.; Maydan, D.; McDonald, J.; Menon, R. *Parallel programming in OpenMP*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2001.

(45) Intel Math Kernel Library for Linux OS User's Guide. http://www.intel.com/software/products/ (accessed Jan 11, 2010).

(46) Liu, P.; Shi, Q.; Hal Daumé, H., III; Voth, G. A. *J. Chem. Phys.* **2008**, *129*, 214114.

(47) Press, W. H.; Teukolsky, S. A.; Vettering, W. T.; Flannery, B. P. *Numerical Recipes in C, The Art of Scientific Computing, Second Edition*; Cambridge University Press: New York, NY, USA, 1997.

(48) Hansen, P. C. "The L-curve and Its Use in the Numerical Treatment of Inverse Problems"; in Computational Inverse Problems in Electrocardiology, ed. P. Johnston, Advances in Computational Bioengineering, 2000.

(49) Hansen, P. C.; O'Leary, D. P. *SIAM J. Sci. Comput.* **1993**, *14*, 1487.

(50) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. *J. Chem. Phys.* **1983**, *79*, 926.

CT900643R