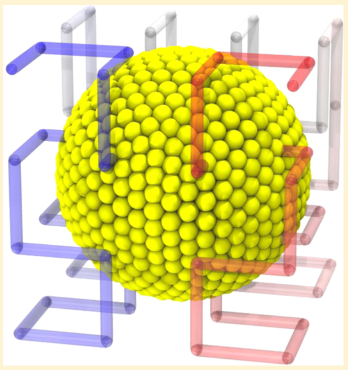
Highly Scalable and Memory Efficient Ultra-Coarse-Grained Molecular Dynamics Simulations

John. M. A. Grime and Gregory A. Voth*

Department of Chemistry, James Franck Institute, Institute for Biophysical Dynamics and Computation Institute, University of Chicago, 5735 South Ellis Avenue, Chicago, Illinois 60637, United States

S Supporting Information

ABSTRACT: The use of coarse-grained (CG) models can significantly increase the time and length scales accessible to computational molecular dynamics (MD) simulations. To address very large-scale phenomena, however, requires a careful consideration of memory requirements and parallel MD load balancing in order to make efficient use of current supercomputers. In this work, a CG-MD code is introduced which is specifically designed for very large, highly parallel simulations of systems with markedly non-uniform particle distributions, such as those found in highly CG models having an implicit solvent. The CG-MD code uses an unorthodox combination of sparse data representations with a Hilbert space-filling curve (SFC) to provide dynamic topological descriptions, reduced memory overhead, and advanced load-balancing characteristics. The results of representative large-scale simulations indicate that our approach can offer significant advantages over conventional MD techniques, and should enable new classes of CG-MD systems to be investigated.



1. INTRODUCTION

Computational molecular dynamics (MD)[1−4] simulations use a discrete time step to numerically integrate molecular equations of motion. MD has demonstrated great flexibility in the study of, for example, biomolecular systems such as lipid membranes,[5−8] proteins,[9−13] and nucleic acids.[14−17] Atomic-level representations are computationally expensive and hence are routinely employed only for MD systems on the order of millions of atoms, with time and length scales of tens of nanoseconds and nanometers (although the upper limits continue to increase).[18,19] Removing selected degrees of freedom from a target molecular system to produce "coarse-grained" (CG) representations[20−23] can significantly extend the reach of computer simulations, particularly where the effects of explicit solvent molecules (such as water in a biological system) are instead approximated by an implicit solvent approach.[24−26] The use of CG models can potentially avoid time-consuming Fourier-space treatments of the electrostatic potential energy,[27] with such effects represented instead by screened interactions with only real space components[28,29] as the lower CG particle number densities make the use of a larger interaction cutoff distance practical.

CG molecular models are computationally efficient, but the application of such models to very large length-scale processes can present a significant challenge even to modern supercomputing resources. The effective use of parallel supercomputers requires an even spread of calculations across separate compute nodes, and this requirement is particularly important for MD simulations: calculating the forces acting on a molecule may require information from several different compute nodes, and if any node is slow to provide such information, then the simulation halts until the appropriate data is coalesced. Parallel MD performance is therefore limited to a function of the "slowest" compute node at each integration step, with imbalances in the assigned MD workloads likely to result in systemic performance degradation.

Many MD codes (with equally variable performance characteristics) are publically available, with NAMD,[30] GROMACS,[31] LAMMPS,[32] CHARMM,[33] HOOMD-blue,[34] and Desmond[35] perhaps the most popular in the wider scientific community. In principle, such codes are applicable to near-arbitrary systems, but the traditional MD focus has rested on relatively small length scales, atomic or near-atomic levels of detail, and broadly uniform particle distributions—with design choices reflecting these priorities. We instead present here a description of the design and performance characteristics of a CG-MD simulation program intended specifically for dynamic, very large-scale simulations with wide inhomogeneities in particle distribution. Particular attention is devoted to issues of memory use and load balancing on modern compute clusters. Although the design of the CG-MD code is inspired by the study of biomolecular phenomena with implicit solvent, the techniques described will have wide applicability in other fields such as computational materials science. Performance is compared to LAMMPS,[32] a well-established conventional MD archetype whose flexibility and ease of modification are attractive in the context of CG simulations. The choice of the LAMMPS MD code allows the straightforward use of an identical compiler and MPI communications library, and hence

Received: August 14, 2013
Published: November 5, 2013

provides the most direct comparison of the CG-MD algorithms to conventional approaches (as implemented in LAMMPS) when running on identical compute hardware.

2. METHODS

2.1. Overview of the CG-MD Algorithm. Large-scale, implicit solvent CG-MD simulations may feature very inhomogeneous distributions of billions of particles across large spatial volumes. Under these conditions, limiting memory requirements and balancing the MD workload across large numbers of compute nodes is critically important. Rather than reimplement common MD algorithms, we instead take this opportunity to explore more unorthodox techniques toward the aim of enabling dynamic, memory-efficient CG simulations at very large scales. Although the use of hardware-specific functionality can increase MD code performance, we avoid such extensions where possible to ensure portability to different compute platforms. All techniques described in this work are therefore implemented in standard C++ using functionality in version 2 of the standard MPI communications libraries[36] (except where explicitly noted) in ca. 6000 lines of source code.

A basic knowledge of conventional parallel MD algorithms is assumed, as described by the numerous standard texts,[1−4] and the following nomenclature is defined for clarity: a node n_i is one of the N_{nodes} instances of the CG-MD code in a parallel simulation; a *subdomain* is a spatial region associated with a particular node; a particle p_i is local to n_i where n_i integrates p_i's equations of motion (i.e., n_i "owns" p_i); a particle p_i is a ghost on n_i where p_i interacts with particles local to n_i but is not itself local (i.e., information on p_i must be shared with n_i by another node); *peers* are nodes which communicate directly, for example, to share ghost particle information for interactions across node boundaries or in the diffusive migration of local particles between nodes.

We adopt the convention of using a limited "skin" distance in addition to the main non-bonded interaction cutoff length, with Verlet lists updated when any particle has traveled more than half of the skin distance. Verlet list updates also trigger the migration of local particles between nodes, the rebuilding of ghost particle buffers, and the regeneration of local sparse data structures.

The basic components of the CG-MD code can be divided under two main headings: sparse data structures (to reduce memory use, described in sections 2.2 and 2.3) and load balancing/communications (for effective use of parallel compute resources, sections 2.4 and 2.5).

2.2. Sparse Dynamic Molecular Topologies. Molecular structure is preserved in MD simulations with the use of topological connections between particles, such as covalent bonds or valence angles. An explicit global listing of such connectivity, with bonds and angles, etc., defined in terms of unique particle identifiers, allows straightforward calculation of local topological forces even as particles migrate between nodes at runtime. Explicit global topology lists are simple and convenient, but they can be inflexible and waste resources—particularly where the global topology list is duplicated across all nodes and many molecules are effectively repeating the same fundamental structure. In the interests of a more flexible and efficient MD code, we eschew the usual global topology lists to instead define connectivity using template molecules.

A template molecule describes molecular topology using particle offsets into a template structure. If particles in the simulation are tagged with a molecule identifier, template molecule type, and local index into the template, it is possible to reconstruct dynamically the appropriate local topology while storing only a single copy of the template. One immediate advantage to the use of template molecules is a significant reduction in the complexity and size of simulation input files. For an example system of 1×10^6 CG lipid molecules, where each CG lipid is composed of five CG particles with four covalent bonds and three valence angles, the topological information required by the LAMMPS MD code occupies approximately 240 MB of disk space. The size of this global topological listing increases as a monotonic function of the number of CG lipids present. By contrast, the same topological information requires approximately 500 bytes in the present CG-MD code, and this description remains valid regardless of the number of CG lipids in the simulation. Template molecules can be combined with a limited number of explicit topological entries to describe, for example, connected chains of template subunits while retaining the essential low-memory advantages of implicit topological descriptions.

The use of template molecules suggests other intriguing functionalities, for example, elastic network models (ENMs)[37−40] of a protein dynamically switching between two stable conformations in response to local conditions. Where the two distinct structures are modeled with separate template molecules, it becomes a simple matter to alternate between conformations at runtime by switching the template type at runtime and invoking local topology regeneration. Only slightly more complicated are models that involve anabolic or catabolic reactions, with reactants and products described using separate template molecules whose types are switched at runtime to simulate the creation or removal of bonding information. These phenomena can be difficult to model in the presence of a full global topology list: where many instances of such molecules are present in the simulation, and changes are a dynamic function of local environment, managing the global topology listing may be nontrivial. The sparse, dynamic topological descriptions make the CG-MD code a natural complement to the use of "ultra-coarse-grained" models.[41]

2.3. Sparse Spatial Data Structures. The calculation of non-bonded interactions in an MD simulation typically uses Verlet lists[42] generated with a link cell algorithm.[1] Link cell algorithms divide the spatial volume of interest into a notional lattice of cells, with lattice spacing $\geq r_{\text{cut}}$, the non-bonded interaction length. By mapping lists of local particles into their enclosing link cells (a very efficient operation), the contents of cells in the immediate vicinity of a particle p may be examined to determine which other particles should be included in p's Verlet list. This technique significantly reduces the number of particles that must be considered when building a Verlet list, improving simulation performance. Further efficiency gains may be possible by subdividing the link cells, reducing the number of particles tested at the cost of increasing the number of link cells which must be processed.[1]

Conventional link cell approaches require memory proportional to the simulated volume, and therefore proportional to a cubic function of some characteristic local subdomain length L. For the relatively small L values present in all-atom MD simulations, this memory overhead is almost negligible. For an MD code designed to enable cell-scale simulations, where L may be on the order of micrometers or more, the link cell memory overheads warrant more careful consideration. A simple motivation is shown in Figure 1, where the memory required for a single-particle simulation using the LAMMPS

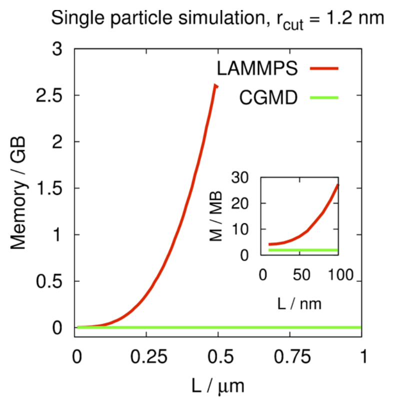


Figure 1. Memory use as a function of simulation box length L for the same single-particle system in both LAMMPS (red curve) and the CG-MD code (green curve). All simulations are run on a single CPU with r_{cut} = 1.2 nm. Truncation of the LAMMPS curve at $L \approx 0.5$ μm indicates the onset of paging to virtual memory, with catastrophic performance effects.

MD code is plotted as a function of L (red curve). Although the simulation described by Figure 1 is almost totally empty, the memory overhead becomes very large as L increases with constant link cell size.

The present CG-MD code uses a "sparse" link cell algorithm to prevent this behavior (Figure 1, green curve). For cell-scale CG-MD simulations with implicit solvent, relatively large regions of the simulated volume may be effectively empty. Rather than populate the entire simulated volume with link cells, regardless of whether they are occupied, we instead use a data structure that maps the lattice positions of occupied link cells (integer x, y, z triplets) onto the list of particles contained in the link cell. The sparse link cell map uses no memory for empty regions of space, and is regenerated immediately prior to Verlet list rebuilds. Assuming a notation where N_{p} denotes the number of particles, N_{conv} is the number of conventional link cells, and N_{sparse} is the number of sparse link cells, we may draw the following conclusions: where the N_{p} particles have a uniform distribution in a relatively small volume, such that $N_{\text{p}} \geq N_{\text{conv}}$, N_{sparse} is strictly equal to N_{conv}. However, where $N_{\text{p}} \ll N_{\text{conv}}$, as can be the case for cell-scale CG simulations with implicit solvent, N_{sparse} is strictly $\leq N_{\text{p}}$ because each sparse link cell may be occupied by more than one particle. The memory needed by the sparse link cells is therefore proportional to the smaller of N_{p} or N_{conv}, decoupling link cell memory requirements from the need to cover empty regions of the simulation volume (Figure 1). This technique enables implicit solvent CG simulations of very large volumes in a memory efficient manner.

2.4. Flexible, Work-Balanced Spatial Decomposition.

Traditional parallel MD algorithms divide the total simulated volume into subdomains,[32,30,31] with any calculations required inside a subdomain performed by a particular CPU core, although variations exist where the subdomain on which particles reside does not necessarily perform the associated local MD calculations (such as IBM's *Blue Matter*[43] and D. E. Shaw Research's *Desmond* code[44]). Subdomains are generally arranged as stacked parallelepipeds for simple organization and communication, for example, when sharing particle information for non-bonded interactions across node boundaries or where diffusion leads to particle migration between nodes. If a uniform distribution of particles exists throughout the entire simulation, basic load balancing naturally emerges from a uniform decomposition of the total simulated volume (Figure 2a).

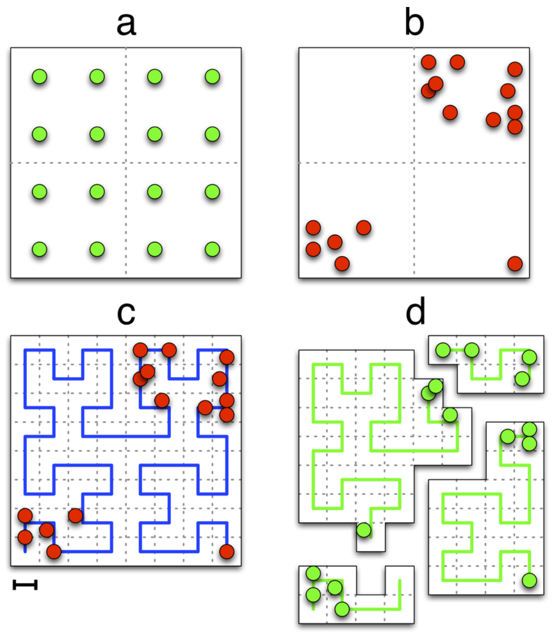


Figure 2. 2-D examples of spatial decomposition, assuming four CPUs. Uniform particle distributions are simple to load balance (a, dashed lines denote subdomains), but non-uniform distributions are unbalanced under the same decomposition (b). Applying a Hilbert SFC to the system (c, lattice spacing denoted with horizontal bar) allows division of the SFC into sections containing approximately equal particle counts (d, SFC sections separated for clarity).

For simulations with pronounced local density variations, such as might be expected for CG models with implicit solvent, uniform domain decomposition can produce serious load imbalances. A simple example of this phenomenon is shown in Figure 2b, where the aggregation of particles creates a large empty region with localized particle clusters. In this situation, the bulk of the MD workload is performed by a relatively small number of CPUs, to the obvious detriment of simulation performance. One possible solution is to adjust the dividing planes that delineate subdomains,[31] retaining the basic stacked parallelepiped organization but altering the volume (and, hence, the number of particles) contained in each subdomain. More exotic schemes may further partition the local subdomain calculations into independent work units.[30] We instead load balance on the basis of a Hilbert space-filling curve (SFC), with the SFC implementation based on the algorithms of J. K. Lawder.[45,46] Space filling curves have been used in computational astrophysics (e.g., ChaNGa[47] and GAMER[48]), which provided the inspiration for this CG-MD code, but are not unknown in MD simulation: HOOMD-blue[34] uses a Hilbert SFC to reorder local particles for more efficient memory access, and versions of NAMD use SFCs to generate initial static mappings of CPUs to communication objects to optimize the use of network resources.[49]

A detailed description of the mathematics of space filling curves is beyond the scope of this work, and the curious reader should consult more specialized texts. In simple terms, the Hilbert SFC allows conversion of higher dimensional coordinates into 1-D SFC indices that preserve locality: if two particles are local in 3-D Cartesian space, their 1-D SFC indices are likewise proximate. Hilbert SFC vertices connect the centers of a notional lattice of cells (Figure 2c), and by mapping particle coordinates into these cells, the SFC can be divided into contiguous sections containing approximately equal numbers of particles (Figure 2d). Where each node calculates the particle counts for locally occupied cells, the sets of local cell indices and populations may then be transmitted to a specified load-balancing node that sections the SFC as appropriate. The resultant set of SFC sections (and hence the spatial subdomains defined by each SFC section) is then broadcast to all nodes, followed by particle migration to ensure nodes contain only those particles appropriate to their newly assigned subdomains.

In principle, the Hilbert SFC load balancer may partition the simulation into a regular cuboidal lattice if uniform particle density exists throughout the system and the number of CPU cores assigned to the simulation is equal to the total number of Hilbert SFC cells present. Under such conditions, the Hilbert SFC domain layout can resemble that of more conventional MD spatial decomposition algorithms. Depending on the precise distribution of particles and the number of assigned CPU cores, it is possible to partition workloads into regular lattices with different lattice spacing, or even a set of parallelepiped domains of differing sizes; such arrangements would, however, be rather sensitive to the exact distribution of particles upon invoking the load balancer and are unlikely to be common occurrences.

The use of a Hilbert SFC cell as the fundamental unit of load balancing may also take into account, e.g., the bonds and angles contained by each SFC cell, although here we restrict ourselves to considering local particle counts as a proxy to estimate the true workload: evaluating bonded and non-bonded forces, with MD simulation time typically dominated by the latter. A variation of this approach involves nodes also counting the number of particles in the 3 × 3 × 3 SFC cell neighborhood surrounding each occupied cell, to more directly estimate the non-bonded interaction workloads. This approach did not offer noticeably superior performance for the systems tested, but such a treatment may be valuable in other situations.

2.5. Dynamically Networked, Latency Tolerant Internode Communication. The sections of the Hilbert SFC assigned by the dynamic load balancer can describe spatial subdomains with very irregular boundaries. This requires an equally dynamic approach to internode communications, as nodes can no longer rely on a static and well-organized arrangement of neighboring subdomains. By setting the notional Hilbert SFC lattice spacing to be $\geq r_{\text{cut}}$, this process becomes straightforward: each node simply examines the 3 × 3 × 3 set of SFC cells which surround each local particle, detecting cells which lie in a remote domain (Figure 3, left panel), and generating a map of remote node identifiers ("peers") to the lists of particles which may be of interest. It is then simple to migrate particles between nodes or share ghost particle information by transmitting appropriate particle data as indicated by the local peer map. A notional Hilbert SFC lattice spacing $\geq r_{\text{cut}}$ combined with testing 3 × 3 × 3 SFC cell neighborhoods ensures that particles with non-bonded interactions across node boundaries are shared as ghosts, analogous to link cell algorithms in the generation of Verlet lists.

It is important to note that, although the dynamic peer mapping approach allows node n_i to detect a new peer node n_j,

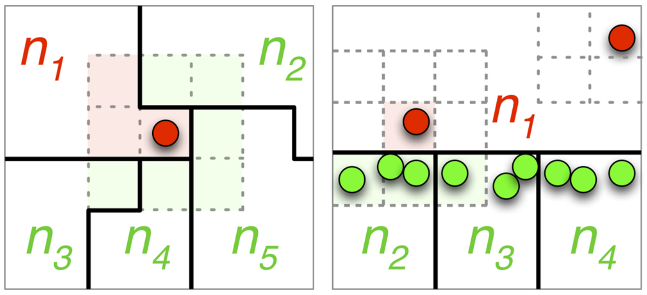


Figure 3. 2-D illustration of the dynamic communication mapping and data filtering as described in the text. Left panel: The set of cells enclosing a particle on node n_1 (red circle) are examined, with some found to lie in the subdomains of n_{2-5} (green squares), triggering communication between n_1 and n_{2-5}. Right panel: Nodes n_{2-4} contain particles close to the subdomain of n_1, but only a subset these particles interact with particles local to n_1. Nodes n_{2-4} transmit the lattice coordinates of cells containing these particles to n_1, which returns a filtered list of only those cells whose contents can interact with particles local to n_1 (green squares). Node n_1 likewise transmits the lattice coordinates of the occupied cell in the border region (highlighted in red) to be examined by n_2 and n_3. The contents of two cells in n_2 and one cell in n_3 will therefore be shared with n_1 as ghosts (green squares), and the contents of a single cell on n_1 (red squares) shared with n_2 and n_3; no ghosts are shared between n_1 and n_4. Subdomain boundaries are solid lines, cells dashed lines.

n_j does not automatically know to listen to n_i for incoming data. A simple solution to this problem is a regular global update of peer information, for example, using the command *MPI_Alltoall()*. This approach, however, was found to be inefficient for large N_{nodes}, so we instead use remote memory access (RMA) operations. Each node exposes a local memory array of N_{nodes} integers to RMA operations, and if (for example) node n_5 needs to begin sharing data with node n_{12}, then n_5 writes a value of 1 directly into element 5 of n_{12}'s exposed memory array. It is then straightforward for all nodes to detect new peers by detecting nonzero values in their local memory arrays, with appropriate synchronization to prevent race conditions. All elements in the local arrays are reset to values of zero after internal data is rebuilt, to ensure dynamic regeneration of the peer mappings at the next rebuild step. RMA operations were introduced in v2 of the MPI communication standard,[36] but poor performance of these routines for large numbers of compute nodes instead led to the use of Cray's network-native DMAPP operations.

As sections of the Hilbert SFC could potentially describe large spatial subdomains, with commensurately large surface areas, there exists the potential for significant wasted communication when sharing ghost particle information across subdomain boundaries. A relatively large subdomain (and hence, surface area) assigned to a node n_i may result in n_i receiving large numbers of ghost particles, even where relatively few of these ghosts actually interact with particles local to n_i. A simple illustration of this problem is shown in Figure 3 (right panel). We therefore prefilter ghost particle transmissions using the lattice coordinates of occupied SFC cells: each node transmits the lattice coordinates of occupied cells of potential interest to remote nodes, and the remote nodes return a list of only those cells whose contents are actually useful. It is then straightforward to transmit only the ghost particle data described by the filtered cell lists, thus reducing unnecessary communication. The full ghost particle information is transmitted in the first MD step after the rebuilding of internal data;

subsequent steps transfer only the updated ghost particle coordinates until the next rebuild is triggered, helping to reduce the quantity of data communicated between peers.

The dynamic peer-mapping algorithm leads most naturally to a single-pass sharing of particle data using asynchronous point-to-point routines as opposed to algorithms with several communications "sweeps" on each Cartesian axis.[32,44,31] It is therefore possible to help mask communication latency using two sets of Verlet lists on each node: the first set contains only pairs of local particles, and the second contains only pairs of local particles and ghost particles. The non-bonded interactions between local particles (which require no information from remote nodes) can then be evaluated while waiting for ghost particle information to be delivered. This approach can be extended to the use of separate Verlet lists for each ghost particle source, further overlapping MD calculations and communication, but this was not found to provide noticeably improved performance for the systems tested. If runtime delays manifest via temporary system buffering or forced rendezvous paths for incoming particle data, the single-pass approach also enables a "double buffering" technique: MPI receives for subsequent iterations of the MD loop can be posted on alternating sources (such as MPI communicators) to ensure that any incoming data always has a matching receive preposted, even if one or more nodes lag the general simulation.

Communication routines in the CG-MD code are thus decoupled from any particular spatial arrangement of peers, providing automatic and highly dynamic runtime detection of emerging data transfer patterns. For example, a node n_i may simply insert a completely new molecule into its local data and invoke the peer mapping routine—after which the molecule will be automatically split and distributed across subdomains as appropriate, even where the new molecule spans multiple subdomains that do not share borders with n_i.

3. TEST SYSTEMS

Recovering the properties of a target system using CG models can require very different representations of the bonded and non-bonded energies, depending on the nature of the target system and the desired level of coarse graining. The computational efficiency of different interaction forms can vary widely, particularly for many-body non-bonded potentials,[50] and relatively expensive interactions may be needed to recapitulate the behaviors of the detailed system. Although a particular interaction type might be unnecessary or prohibitively time-consuming in an all-atom simulation, it may be attractive in the context of a CG model with fewer interacting particles. As an exhaustive consideration of all possible CG interaction forms is obviously infeasible, we will instead describe the CG-MD code performance using a familiar baseline of the venerable Lennard-Jones (LJ) 12−6 potential.

The LJ potential is well-known in the field of molecular modeling, at least in part due to its computational efficiency. The common Gay−Berne potential[51] (which has been used in CG models),[52−56] by comparison, is estimated to require ~15 times the computational work.[57] Parallel MD performance is a balance between the time required for MD force calculation and that of any background communication, etc., so the choice of a lightweight LJ potential should provide a rigorous test of the CG-MD scaling behavior.

Full details of all simulations are provided in the Supporting Information, but here we note one particular aspect: although the non-bonded forces were calculated in each case, these forces were not used in the subsequent integration of the equations of motion. The important benefit to this approach is that identical parallel simulations (generating identical trajectories and data communication) may be run in both the presence and absence of Verlet list generation and non-bonded force calculation. This technique offers a very direct measurement of CG-MD code overheads in excess of the non-bonded interaction workloads that typically dominate MD simulation times. This technique also ensures fast diffusion of the CG particles, providing stringent performance tests of the sparse data structures and communication routines (due to more frequent Verlet list rebuilds, internal data regeneration, and particle migrations).

The initial testing of the CG-MD code is divided into two main sections: the first section describes a series of single CPU core simulations of a Lennard-Jones fluid over a range of number densities, to examine raw serial performance in the absence of implicit topologies and load balancing. The second section involves large parallel simulations of a representative CG biomolecular system with pronounced inhomogeneity in the particle distributions (addition simulations are also described in the Supporting Information). Identical test simulations are performed in both the CG-MD code and LAMMPS, with simulation time and memory use compared.

4. RESULTS AND DISCUSSION

4.1. Single CPU Core Simulations of a Simple Lennard-Jones Fluid.

Although the CG-MD code is designed for parallel simulations of very large and inhomogeneous molecular systems, we first examine performance in situations where the code might be expected to perform most poorly: simulations of a simple LJ fluid on a single CPU core. The load balancing and sparse data algorithms of the CG-MD code offer no advantages under such circumstances, and indeed, the sparse data algorithms should be assumed to degrade performance due to the additional metadata and memory indirection required. These systems can therefore provide a useful indication of worst-case performance characteristics.

A range of Lennard-Jones fluids with number density 0.001 $Å^{-3} \leq \rho \leq 0.035$ $Å^{-3}$ were simulated on a single CPU core, with the runtimes and memory use presented in Figure 4.

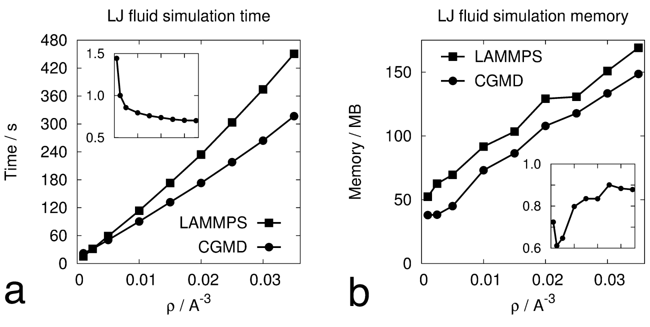


Figure 4. Comparison of simulation time (a) and memory use (b) for identical LJ fluid simulations using LAMMPS and the CG-MD code on a single CPU core. Figures inset: CG-MD simulation time (a) and memory use (b) proportional to that of LAMMPS for the same systems. Horizontal axes are identical to those of the main plots (with labels omitted for clarity). A value of 1.0 on the vertical axis indicates equivalence with LAMMPS, with values >1.0 and <1.0 indicating greater or smaller values than LAMMPS, respectively.

Surprisingly, the CG-MD code displays superior performance to LAMMPS for all but the most dilute LJ fluids, and lower memory use is recorded in all cases. The sparse link cell algorithm introduces a noticeable performance penalty for uniform, very disperse systems (ρ = 0.001 Å^{-3}, ~2 LJ particles per $r_{cut}{}^3$), where Verlet list generation requires roughly twice as long as LAMMPS (~7 s vs ~3.4 s overall, corresponding to ~31% vs ~22% of the total simulation time with the same number of Verlet list rebuilds), but the CG-MD code displays acceptable performance in all other cases. Void-free, uniformly high number density simulations (liquid water, for example, has $\rho \approx 0.033$ Å^{-3}) are diametrically opposed to the intended use of the CG-MD code, and this is therefore a promising result. Although identical LJ particle counts were present in each simulation (125 × 10^3), memory use and simulation time increase as a function of ρ for both MD codes due to the larger numbers of adjacent particle pairs in the Verlet lists at higher ρ.

4.2. Parallel Simulations of an Inhomogeneous CG Biomolecular System. We now consider a system more representative of the intended use of the CG-MD code: simulations of a large, inhomogeneous CG system using a parallel supercomputer. The test system consists of 1000 MD time steps of a model CG lipid bilayer membrane (BLM) of dimensions 2 μm × 2 μm, accompanied by two spherical BLM vesicles of diameter 125 nm (Figure 5, area per lipid of 70 Å^2 for all BLMs). Systems of this type are relevant to, for example, the modeling of HIV virions in the vicinity of a cell membrane[58] or a neurological synapse.[59] The model CG lipids consist of five CG beads connected as a linear polymer via four harmonic bonds and three harmonic valence angles. The total simulation size was 58.5 × 10^6 CG particles, and featured a very uneven distribution of molecular structures in contrast to the simple LJ fluids considered in section 4.1; the highly inhomogeneous particle distributions and computationally inexpensive LJ potentials should provide a stern test of load balancing and communication routines.

Test simulations were performed using the Blue Waters supercomputing resource at the National Center for Supercomputing Applications (NCSA, University of Illinois at Urbana—Champaign). A brief description of the hardware characteristics of this system is warranted, as these aspects influence how the simulations were performed. Each XE compute node on the Blue Waters resource may be considered as four "NUMA domains", each containing a single 8MB L3 cache with four floating point units (FPUs) and eight integer compute cores. Each compute node therefore contains four L3 cache regions, 16 FPUs, and 32 integer CPU cores.

To examine the performance effects of sharing FPUs and L3 cache, we arrange the parallel simulations to use 1, 4, 8, and 16 CG-MD processes per compute node (with each process using a single CPU core). For the case of one and four processes per compute node, each process has a dedicated FPU and L3 cache; for 8 processes per compute node, each has a dedicated FPU but shares a L3 cache with another process; for 16 processes per compute node, each process retains a dedicated FPU but shares L3 cache with three other processes. The test simulation was also run with 31 processes per compute node, such that FPUs are also shared between processes, but a full 32 processes per compute node was not used as core specialization provided background OS processes with a dedicated integer core on each compute node.

Identical simulations were executed on a range of 128 to 16 384 compute nodes, to produce strong scaling tests of the CG-MD code from 128 to ~260 000 CPU cores. The results of these tests are presented in Figure 6a where we observe excellent performance, with all timings in the range of ideal to 85% of ideal scaling and each MD step completing in ≈2.5 ms for the largest CPU core counts. We note again that parallel MD scaling performance is sensitive to the ratio of computational work to communication times, and that the system

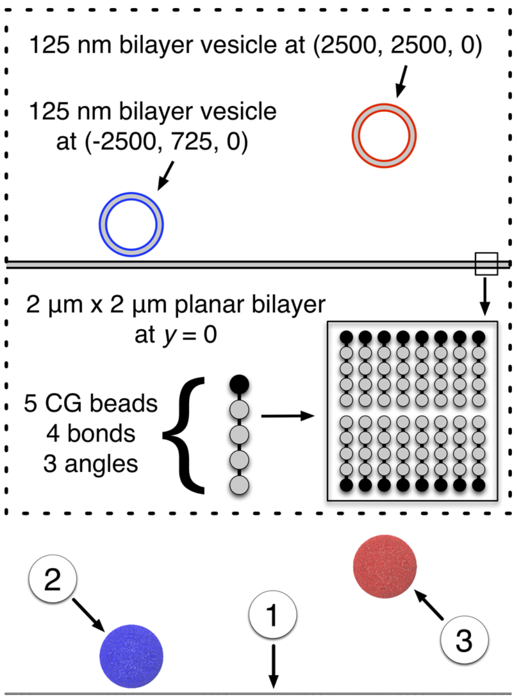

125 nm bilayer vesicle at (2500, 2500, 0)
125 nm bilayer vesicle at (-2500, 725, 0)
2 µm x 2 µm planar bilayer at y = 0
5 CG beads
4 bonds
3 angles
1
2
3

Figure 5. Schematic of the parallel test system as discussed in the main text. Upper panel: two spherical bilayer membranes (BLMs) are positioned above a 2 μm × 2 μm planar BLM (not shown to scale). Lower panel: VMD[60] snapshot of the region of the system containing the planar BLM (1) and the two spherical BLMs (2 and 3). The area per lipid is 70 Å^2 for all BLMs. The periodic simulation cell is 2 μm × 2 μm × 2 μm, with r_{cut} = 2.0 nm.

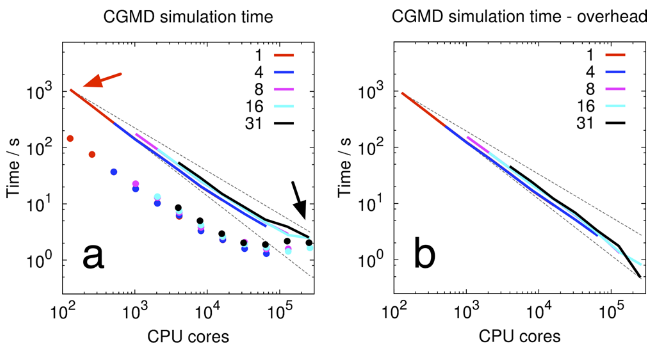


Figure 6. (a) Raw simulation time for 1000 CG-MD time steps of the system described in the text. The line color indicates the number of CG-MD processes per compute node. 128 and ~260 000 CPU cores are indicated with red and black arrows, respectively. Also shown are the timings from identical simulations in the absence of Verlet list generation and non-bonded forces (circles), indicating the CG-MD simulation overheads. (b) Subtracting the overheads from the raw timings indicates the time required to perform non-bonded MD workloads, and hence the load balancer scaling performance. Ideal scaling and 85% of ideal scaling are indicated with broken gray lines.

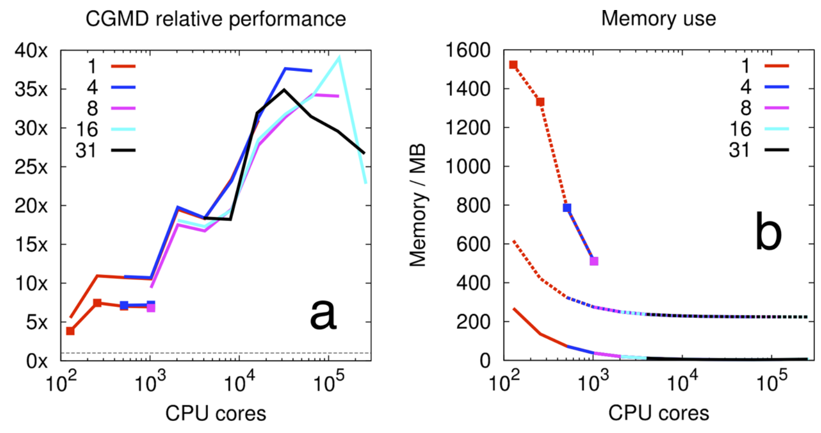


Figure 7. (a) Relative performance of the CG-MD code proportional to LAMMPS with load balancing (lines with squares) and LAMMPS without load balancing (lines). The load-balanced LAMMPS simulations did not run for >1024 CPU cores. Note the similarity in the trend for individual data points from both load-balanced and non-load-balanced LAMMPS simulations. (b) Reported memory use of the CG-MD code (solid lines) compared to LAMMPS with and without load balancing (broken lines with squares and broken lines, respectively).

studied here is extremely computationally lightweight: the average CG particle Verlet list contained ca. 85 entries, in contrast to the approximately 250+ entries that might be expected in a typical all-atom biomolecular simulation, with these non-bonded pair interactions taking the form of a single efficient LJ 12−6 potential. No effort was made to minimize network "hop" distances for internode communications, nor was background network traffic minimized by using an otherwise empty compute resource; these results are therefore likely to indicate "real world" performance. The data point for 31 processes × 16 384 compute nodes (≈500 000 CPU cores) was not available, as the Blue Waters system detected transient errors in the system's DMAPP library and aborted the simulation.

Also shown in Figure 6 are timings for the same simulations without Verlet list generation and non-bonded force calculations (Figure 6a, circles), indicating the background overhead of the CG-MD code in each case. Subtracting the overhead from the raw CG-MD simulation data reveals the time spent in the fundamental MD calculations, as shown in Figure 6b, where we observe almost perfect load balancing performance over 3 logarithmic decades of CPU core counts.

For simulations using ~260 000 CPU cores, the majority of the CG-MD runtime does not involve non-bonded force calculations and the associated Verlet list generation routines (Figure 6a, circles). Bond and angle force calculations comprise only ~5% of this ambient overhead, with the bulk of the time spent waiting for the completion of *MPI_Irecv()* calls for ghost particle data. The use of these MPI routines in the current implementation of the CG-MD code appears to impose a hard limit of ~1000 MD time steps per second, regardless of the precise system simulated (see also Supporting Information 3). This suggests that replacing these MPI routines with appropriate calls to hardware-specific RMA interfaces, such as DMAPP on Cray's *Gemini* interconnect or PAMI for IBM's *Blue Gene* systems, might further improve scaling performance at large CPU core counts.

One interesting aspect of the data presented in Figure 6 is the presence of effectively separate curves in the scaling plots. The first curve features assignments of 1 and 4 CG-MD processes per compute node (i.e., no L3 cache sharing), the second contains 8 and 16 processes per compute node (L3 cache is shared), and the final curve contains 31 processes per compute node (with both L3 cache and FPUs shared). This rather unexpected result indicates the efficiency of the CG-MD simulation engine and communication routines: sharing L3 cache or FPUs affects performance more than distributing the same number of CG-MD processes over a larger number of compute nodes, despite the greater internode communication required. We note that this is unlikely to be an issue of L3 cache starvation, as there is no separation between the timings for 8 and 16 processes per compute node. A similar trend was not clear in the LAMMPS results (see Supporting Information 3.1−3.3).

The relative performance and memory use of the CG-MD code versus LAMMPS for identical simulations are shown in Figure 7. The CG-MD software was between ~5× and ~40× faster than standard LAMMPS, with the performance difference increasing as a function of CPU core count until CG-MD communication overheads limited further improvements. Enabling load balancing in the LAMMPS code improved performance (with the CG-MD code now ~4× to ~7× faster) until LAMMPS halted with errors for >1024 CPU cores. Until that point, the load-balanced LAMMPS simulation timings appear to follow the same general performance trend as that of standard LAMMPS (see also Supporting Information 3.1), and hence, it is likely that the relative performance advantage of the CG-MD code would continue to increase as a function of CPU core count.

The memory use of the CG-MD code was significantly lower than LAMMPS in these test simulations, particularly for load-balanced LAMMPS simulations at lower CPU core counts (Figure 7b), indicating the value of sparse data techniques for large CG simulations. Consideration of the total data presented in Figure 7 suggests that the behavior of the CG-MD code may be divided into two major regimes with respect to LAMMPS: for relatively low CPU core counts, the CG-MD code displays a significant performance improvement of ca. 4−10× alongside a pronounced reduction in memory use. For larger CPU core counts, the difference in memory footprint becomes less important but the CG-MD performance advantage increases up to ~40× for the system discussed here. These characteristics are valuable in the context of cell-scale simulations, avoiding infeasible memory requirements for low CPU counts while providing markedly superior performance with access to larger compute resources. The CG-MD platform described here can therefore offer a "best of both worlds" approach, with the low memory overheads and effective load balancing also suggestive of a solid platform for GPU acceleration.

■ CONCLUSIONS

We have developed and implemented a custom CG-MD code featuring sparse memory techniques to enable dynamic, implicit topology definitions and low memory requirements for cell-scale simulations using implicit solvent CG models. Per-particle work is distributed and parallelized using a Hilbert space-filling curve, with strong scaling demonstrated over hundreds of thousands of CPU cores for a system with very inhomogeneous particle distributions. The code is relatively simple and portable, and is thus amenable to straightforward modification and future extensions. Although the design of the CG-MD code is inspired by the study of biomolecular phenomena, the techniques described are in principle widely applicable in the field of computational materials science, especially for inhomogeneous soft matter systems.

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

Full descriptions of the test simulations described in this manuscript and additional test simulation data. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: gavoth@uchicago.edu.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation through the Center for Multiscale Theory and Simulation (NSF grant CHE-1136709). This research is also part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana—Champaign and its National Center for Supercomputing Applications. This work is additionally part of the "Petascale Multiscale Simulations of Biomolecular Systems" PRAC allocation support by the National Science Foundation (award number OCI 1036184). The authors would like to thank James F. Dama (University of Chicago), Chris Knight and Carolyn Phillips (Argonne National Laboratory), and Robert Brunner (NCSA) for many interesting discussions and valuable advice.

■ REFERENCES

(1) Allen, M. P.; Tildesley, D. J. *Computer simulation of liquids*; Clarendon: Oxford, U.K., 1987.
(2) Leach, A. R. *Molecular modelling: principles and applications*, 2nd ed.; Prentice Hall: Harlow, U.K., 2001.
(3) Frenkel, D.; Smit, B. *Understanding molecular simulation: from algorithms to applications*, 2nd ed.; Academic: San Diego, CA; London, 2002.
(4) Rapaport, D. C. *The art of molecular dynamics simulation*, 2nd ed.; Cambridge University Press: Cambridge, U.K., 2004.
(5) Feller, S. E. Molecular dynamics simulations of lipid bilayers. *Curr. Opin. Colloid Interface Sci.* **2000**, *5*, 217−223.
(6) Benz, R. W.; Castro-Roman, F.; Tobias, D. J.; White, S. H. Experimental validation of molecular dynamics simulations of lipid bilayers: A new approach. *Biophys. J.* **2005**, *88*, 805−817.
(7) Jeon, J. H.; Monne, H. M. S.; Javanainen, M.; Metzler, R. Anomalous Diffusion of Phospholipids and Cholesterols in a Lipid Bilayer and its Origins. *Phys. Rev. Lett.* **2012**, *109*, 188103.
(8) Notman, R.; Anwar, J. Breaching the skin barrier–insights from molecular simulation of model membranes. *Adv. Drug Delivery Rev.* **2013**, *65*, 237−250.
(9) Karplus, M.; McCammon, J. A. Molecular dynamics simulations of biomolecules. *Nat. Struct. Biol.* **2002**, *9*, 646−652.
(10) Karplus, M.; Kuriyan, J. Molecular dynamics and protein function. *Proc. Natl. Acad. Sci. U.S.A.* **2005**, *102*, 6679−6685.
(11) Adcock, S. A.; McCammon, J. A. Molecular dynamics: survey of methods for simulating the activity of proteins. *Chem. Rev.* **2006**, *106*, 1589−1615.
(12) Scheraga, H. A.; Khalili, M.; Liwo, A. Protein-folding dynamics: overview of molecular simulation techniques. *Annu. Rev. Phys. Chem.* **2007**, *58*, 57−83.
(13) Dodson, G. G.; Lane, D. P.; Verma, C. S. Molecular simulations of protein dynamics: new windows on mechanisms in biology. *EMBO Rep.* **2008**, *9*, 144−150.
(14) Lytton-Jean, A. K. R.; Gibbs-Davis, J. M.; Long, H.; Schatz, G. C.; Mirkin, C. A.; Nguyen, S. T. Highly Cooperative Behavior of Peptide Nucleic Acid-Linked DNA-Modified Gold-Nanoparticle and Comb-Polymer Aggregates. *Adv. Mater.* **2009**, *21*, 706.
(15) Hariharan, M.; McCullagh, M.; Schatz, G. C.; Lewis, F. D. Conformational Control of Thymine Photodimerization in Single-Strand and Duplex DNA Containing Locked Nucleic Acid TT Steps (vol 132, pg 12856, 2010). *J. Am. Chem. Soc.* **2010**, *132*, 15831−15831.
(16) Mocci, F.; Laaksonen, A. Insight into nucleic acid counterion interactions from inside molecular dynamics simulations is "worth its salt". *Soft Matter* **2012**, *8*, 9268−9284.
(17) Cheatham, T. E., 3rd; Case, D. A. Twenty-five years of nucleic acid simulations. *Biopolymers* **2013**.
(18) Piana, S.; Lindorff-Larsen, K.; Shaw, D. E. Atomic-level description of ubiquitin folding. *Proc. Natl. Acad. Sci. U.S.A.* **2013**, *110*, 5915−5920.
(19) Zhau, G.; Perilla, J. R.; Yufenyuy, E. L.; Meng, X.; Chen, B.; Ning, J.; Ahn, J. W.; Gronenborn, A. M.; Schulten, K.; Aiken, C.; Zhang, P. Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. *Nature* **2013**, *497*, 643−646.
(20) Nielsen, S. O.; Lopez, C. F.; Srinivas, G.; Klein, M. L. Coarse grain models and the computer simulation of soft materials. *J. Phys.: Condens. Matter* **2004**, *16*, R481−R512.
(21) *Coarse-graining of condensed phase and biomolecular systems*; Voth, G. A., Ed.; CRC Press: Boca Raton, FL, 2009.
(22) Hadley, K. R.; McCabe, C. Coarse-Grained Molecular Models of Water: A Review. *Mol. Simul.* **2012**, *38*, 671−681.
(23) Saunders, M. G.; Voth, G. A. Coarse-graining methods for computational biology. *Annu. Rev. Biophys.* **2013**, *42*, 73−93.
(24) Chen, J.; Brooks, C. L., 3rd; Khandogin, J. Recent advances in implicit solvent-based methods for biomolecular simulations. *Curr. Opin. Struct. Biol.* **2008**, *18*, 140−148.
(25) Feig, M.; Tanizaki, S.; Sayadi, M. Chapter 6 Implicit Solvent Simulations of Biomolecules in Cellular Environments. In *Annual Reports in Computational Chemistry*; Ralph, A. W., David, C. S., Eds.; Elsevier: 2008; Vol. *4*, pp 107−121.
(26) Onufriev, A., Chapter 7 Implicit Solvent Models in Molecular Dynamics Simulations: A Brief Overview. In *Annual Reports in Computational Chemistry*; Ralph, A. W., David, C. S., Eds.; Elsevier: 2008; Vol. *4*, pp 125−137.
(27) Toukmaji, A. Y.; Board, J. A. Ewald summation techniques in perspective: A survey. *Comput. Phys. Commun.* **1996**, *95*, 73−92.
(28) DeMille, R. C.; Cheatham, T. E., 3rd; Molinero, V. A coarse-grained model of DNA with explicit solvation by water and ions. *J. Phys. Chem. B* **2011**, *115*, 132−142.

(29) Ramachandran, A.; Guo, Q.; Iqbal, S. M.; Liu, Y. Coarse-grained molecular dynamics simulation of DNA translocation in chemically modified nanopores. *J. Phys. Chem. B* **2011**, *115*, 6138−6148.

(30) Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kale, L.; Schulten, K. Scalable molecular dynamics with NAMD. *J. Comput. Chem.* **2005**, *26*, 1781−1802.

(31) Hess, B.; Kutzner, C.; van der Spoel, D.; Lindahl, E. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.* **2008**, *4*, 435−447.

(32) Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular-Dynamics. *J. Comput. Phys.* **1995**, *117*, 1−19.

(33) Brooks, B. R.; Brooks, C. L.; Mackerell, A. D.; Nilsson, L.; Petrella, R. J.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; Caflisch, A.; Caves, L.; Cui, Q.; Dinner, A. R.; Feig, M.; Fischer, S.; Gao, J.; Hodoscek, M.; Im, W.; Kuczera, K.; Lazaridis, T.; Ma, J.; Ovchinnikov, V.; Paci, E.; Pastor, R. W.; Post, C. B.; Pu, J. Z.; Schaefer, M.; Tidor, B.; Venable, R. M.; Woodcock, H. L.; Wu, X.; Yang, W.; York, D. M.; Karplus, M. CHARMM: The Biomolecular Simulation Program. *J. Comput. Chem.* **2009**, *30*, 1545−1614.

(34) Anderson, J. A.; Lorenz, C. D.; Travesset, A. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.* **2008**, *227*, 5342−5359.

(35) Bowers, K. J.; Chow, E.; Xu, H.; Dror, R. O.; Eastwood, M. P.; Gregersen, B. A.; Klepeis, J. L.; Kolossváry, I.; Moraes, M. A.; Sacerdoti, F. D.; Salmon, J. K.; Shan, Y.; Shaw, D. E. In *Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters*, Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Tampa, FL, Nov 11-17; ACM, New York: Tampa, FL, 2006.

(36) Gropp, W.; Lusk, E.; Thakur, R. *Using MPI-2: advanced features of the message-passing interface*; MIT Press: Cambridge, MA, 1999.

(37) Tirion, M. M. Large Amplitude Elastic Motions in Proteins from a Single-Parameter, Atomic Analysis. *Phys. Rev. Lett.* **1996**, *77*, 1905−1908.

(38) Bahar, I.; Atilgan, A. R.; Erman, B. Direct evaluation of thermal fluctuations in proteins using a single-parameter harmonic potential. *Folding Des.* **1997**, No. 3, 173−181.

(39) Atilgan, A. R.; Durell, S. R.; Jernigan, R. L.; Demirel, M. C.; Keskin, O.; Bahar, I. Anisotropy of fluctuation dynamics of proteins with an elastic network model. *Biophys. J.* **2001**, *80*, 505−515.

(40) Lyman, E.; Pfaendtner, J.; Voth, G. A. Systematic multiscale parameterization of heterogeneous elastic network models of proteins. *Biophys. J.* **2008**, *95*, 4183−4192.

(41) Dama, J. F.; Sinitskiy, A. V.; McCullagh, M.; Weare, J.; Roux, B.; Dinner, A. R.; Voth, G. A. The Theory of Ultra-Coarse-Graining. 1. General Principles. *J. Chem. Theory Comput.* **2013**, *9*, 2466−2480.

(42) Verlet, L. Computer "Experiments" on Classical Fluids. 1. Thermodynamical Properties of Lennard-Jones Molecules. *Phys. Rev.* **1967**, *159*, 98−103.

(43) Fitch, B. G.; Rayshubskiy, A.; Eleftheriou, M.; Ward, T. J. C.; Giampapa, M. E.; Pitman, M. C.; Pitera, J. W.; Swope, W. C.; Germain, R. S. Blue Matter: Scaling of N-body simulations to one atom per node. *IBM J. Res. Dev.* **2008**, *52*, 145−158.

(44) Bowers, K. J.; Dror, R. O.; Shaw, D. E. The midpoint method for parallelization of particle simulations. *J. Chem. Phys.* **2006**, *124*, 18.

(45) Lawder, J. K. The application of space-filling curves to the storage and retrieval of multi-dimensional data, Ph.D. Thesis, University of London: London, 1999.

(46) Lawder, J. K.; King, P. J. H. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record* **2001**, *30*, 19−24.

(47) Jetley, P.; Gioachin, F.; Mendes, C.; Kale, L. V.; Quinn, T. Massively parallel cosmological simulations with ChaNGa. *2008 Ieee International Symposium on Parallel & Distributed Processing* **2008**, *Vols. 1−8*, 1945−1956.

(48) Shukla, H.; Schive, H.-Y.; Woo, T.-P.; Chiueh, T. In *Multi-science applications with single codebase - GAMER - for massively parallel architectures*, SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, Apr 1−15; IEEE: Seattle, WA, 2011.

(49) Mei, C. L.; Sun, Y. N.; Zheng, G.; Bohm, E.; Kale, L. In *Enabling and Scaling Biomolecular Simulations of 100 Million Atoms on Petascale Machines with a Multicore-optimized Message-driven Runtime*, International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing) 2011, Seattle, WA, Nov 12−18; Seattle, WA, 2011.

(50) Plimpton, S. J.; Thompson, A. P. Computational aspects of many-body potentials. *Mater. Res. Bull.* **2012**, *37*, 513−521.

(51) Gay, J. G.; Berne, B. J. Modification of the Overlap Potential to Mimic a Linear Site-Site Potential. *J. Chem. Phys.* **1981**, *74*, 3316−3319.

(52) Whitehead, L.; Edge, C. M.; Essex, J. W. Molecular dynamics simulation of the hydrocarbon region of a biomembrane using a reduced representation model. *J. Comput. Chem.* **2001**, *22*, 1622−1633.

(53) Golubkov, P. A.; Ren, P. Y. Generalized coarse-grained model based on point multipole and Gay-Berne potentials. *J. Chem. Phys.* **2006**, *125*, 064103.

(54) Ayton, G. S.; Voth, G. A. Hybrid Coarse-Graining Approach for Lipid Bilayers at Large Length and Time Scales. *J. Phys. Chem. B* **2009**, *113*, 4413−4424.

(55) Orsi, M.; Michel, J.; Essex, J. W. Coarse-grain modelling of DMPC and DOPC lipid bilayers. *J. Phys.: Condens. Matter* **2010**, *22*, 155106.

(56) Wu, J.; Zhen, X.; Shen, H. J.; Li, G. H.; Ren, P. Y. Gay-Berne and electrostatic multipole based coarse-grain potential in implicit solvent. *J. Chem. Phys.* **2011**, *135*, 155104.

(57) Brown, W. M.; Wang, P.; Plimpton, S. J.; Tharrington, A. N. Implementing molecular dynamics on hybrid high performance computers - short range forces. *Comput. Phys. Commun.* **2011**, *182*, 898−911.

(58) Briggs, J. A. G.; Wilk, T.; Welker, R.; Krausslich, H. G.; Fuller, S. D. Structural organization of authentic, mature HIV-1 virions and cores. *EMBO J.* **2003**, *22*, 1707−1715.

(59) Hu, Y. M.; Qu, L.; Schikorski, T. Mean Synaptic Vesicle Size Varies Among Individual Excitatory Hippocampal Synapses. *Synapse* **2008**, *62*, 953−957.

(60) Humphrey, W.; Dalke, A.; Schulten, K. VMD: Visual molecular dynamics. *J. Mol. Graphics Modell.* **1996**, *14*, 33−38.