Title: Petascale Multiscale Simulations of Biomolecular Systems (PRAC sub-award)

Team Lead: Gregory A. Voth [1,2]

Other Personnel: John M. A. Grime [2]

Affiliation: [1] Dept. of Chemistry, Institute for Biophysical Dynamics, James Franck Institute and Computation Institute, University of Chicago, 5735 S. Ellis Avenue, Chicago IL 60439
[2] Computation Institute, Argonne National Laboratory, Lemont IL 60439

Introduction

Computational molecular dynamics[1-4] (MD) with atomic levels of detail can provide valuable insight into processes that are difficult to study with conventional experiments. All-atom MD can be very computationally expensive, however, limiting its utility when considering very large biomolecular systems. Coarse-grained (CG) models instead use simpler representations of a target system, capturing the essential physics while significantly extending the reach of computer simulations - particularly where the effects of explicit solvent molecules (such as water in a biological system) are instead approximated with an implicit solvent.

The application of CG models to cell-scale biological processes remains a significant challenge, even with modern supercomputing resources. Calculating the forces acting on a particular CG molecule may require information from several non-local compute nodes, with the simulation unable to proceed until the required data has arrived. The slowest compute node at each integration timestep thus limits the overall performance of an MD simulation, and hence the ability to "load balance" is of paramount importance for large-scale applications of implicit solvent CG models.

We present the design and initial performance characteristics of a CG-MD simulation program designed specifically for cell-scale implicit solvent CG-MD simulations. Although the motivation for the software lies in biomolecular simulation, it is in principle widely applicable in the field of computational materials science. Rather than simply reproduce existing MD algorithms, we instead explore more unorthodox techniques to enable dynamic, scalable and memory-efficient simulations at very large scales. LAMMPS[5], a mature and well-established MD archetype, is used as a reference code for performance comparisons.

1. Overview of the CG-MD software

The following discussion assumes basic knowledge of conventional parallel MD algorithms as described by numerous standard texts [1-4].

1.1. Sparse molecular topologies

Given a global, explicit listing of all molecular connectivity in the system, calculating local topological information is straightforward even as particles migrate between nodes at runtime. Explicit global topology lists are convenient, but they are also inflexible and waste resources; any local changes to the simulation content must be propagated consistently into the global topology data and multiple instances of the same basic molecule introduce redundant data, as each molecule effectively repeats the same basic structure. In the interests of a more flexible and efficient MD code, we eschew global topology lists to instead define connectivity using template molecules.

A template molecule is an exemplar of a particular type of molecule in the simulation that allows local topology to be inferred dynamically when particle data references a template. Only a single copy of each template is required, regardless of the actual number of molecules corresponding to that template. One immediate advantage to such an approach is the significant reduction of complexity and size for the simulation input files: for example, a system of 1×10^6 CG lipid molecules* requires approximately 240 MB of disk space for the LAMMPS MD code. The size of this global topological listing increases as a monotonic function of the number of CG lipids in the simulation. The same topological description requires approximately 350 bytes in the CG-MD code, with this information valid for any number of CG lipids.

Template molecules enable highly dynamic simulation contents, providing a simple method of altering molecular topologies in response to local environmental conditions and allowing straightforward addition and removal of molecules at runtime. These operations are significantly more complicated when using a global topology list.

1.2. Sparse spatial data structures

The calculation of nonbonded interactions in an MD simulation typically uses Verlet lists[6] generated with a link cell algorithm[1]. Conventional link cell approaches require memory proportional to the simulated volume, and therefore proportional to a cubic function of a characteristic local domain length L. For cell-scale simulations, where L may be on the order of microns or more, link cell memory overheads can be non-trivial. A simple example is shown in **Fig. 1**, where the memory required for a single-particle simulation using the LAMMPS MD code is plotted as a function of L (red curve). Although the simulation described by **Fig. 1** is almost totally empty, the memory overhead becomes very large as L increases.

* Assuming a CG lipid is composed of five CG particles, four covalent bonds and three valence angles

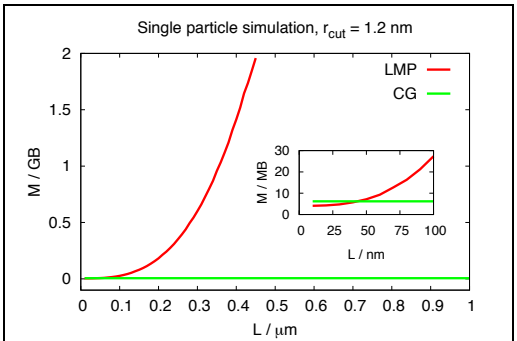

Figure 1: Memory use vs simulation box length L for a single-particle system in LAMMPS (red curve) and the CG-MD code (green curve). All data are single-CPU runs with r_{cut} = 1.2 nm. Truncation of LAMMPS curve at L ≈ 0.45 μm indicates onset of paging to virtual memory, destroying performance.

The CG-MD code uses a "sparse" link cell algorithm to avoid this behavior (**Fig. 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 fill the simulated volume with link cells, regardless of whether they are occupied, we instead use a data structure that maps an integer triplet (the lattice coordinates of an occupied link cell) to the list of particles contained in the link cell. The sparse link cell map therefore uses no memory for empty regions of space, and is regenerated immediately prior to Verlet list rebuilds, thus decoupling link cell memory requirements from simulation volume (**Fig. 1**). This technique enables implicit solvent CG simulations of very large volumes in a memory efficient manner.

1.3. Load balancing

Traditional parallel MD algorithms divide the total simulated volume into subdomains[5,7,8], with calculations inside a subdomain assigned to a specific compute process (hereafter referred to as a "node" of the CG-MD simulation). The subdomains are typically arranged as stacked parallelepipeds for simple organization and communication, for example when sharing particle information across subdomain boundaries.

If a uniform distribution of particles exists throughout the entire simulated volume, basic load balancing naturally emerges from splitting the total volume into uniform subdomains (**Fig. 2 a**). For simulations with pronounced local density variations, such as CG models with implicit solvent, uniform domain decomposition can produce serious load imbalances. A simple example of this phenomenon is shown in **Fig. 2 b**, where CG particles have aggregated to produce a large empty region and local high-density clusters. In this situation, many nodes may be idle while most of the MD workload is performed by a relatively small number of nodes – to the obvious detriment of simulation performance. One possible solution is to adjust the dividing planes that delineate subdomains[7], retaining the basic stacked parallelepiped organization but altering the volume (and hence, the number of

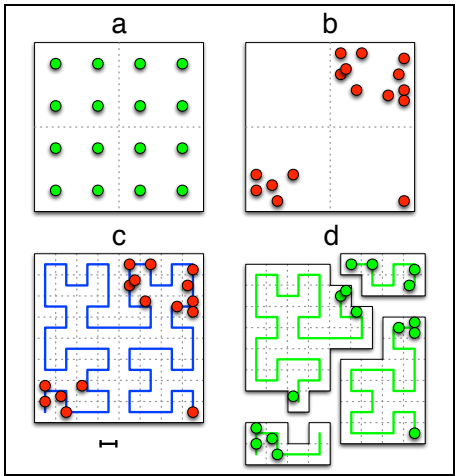

Figure 2: 2-D spatial decomposition using four CPUs. Uniform particle distributions are simple to load balance with equal subdomains (**a**), but non-uniform distributions produce load imbalance (**b**). A Hilbert SFC (**c**, notional lattice spacing as horizontal bar) divides the system into sections of roughly equal particle counts (**d**, sections separated for clarity).

particles) contained in each subdomain. More exotic schemes may further partition the local subdomain calculations into independent work units[8]. We instead load balance on the basis of a Hilbert space-filling curve (SFC).

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[9,10]. The Hilbert SFC converts 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 (**Fig. 2 c**), and by mapping particle coordinates into these cells the SFC can be divided into contiguous sections containing approximately equal numbers of particles (**Fig. 2 d**). SFC sections (and hence the spatial volumes defined by each section) may then be dynamically assigned to specific nodes, allowing runtime load balancing of MD calculations.

1.4. Communications

Sections of a Hilbert SFC can describe subdomains with very irregular boundaries. Dynamic assignment of SFC sections requires an equally dynamic approach to inter-node communications, for example in the sharing of "ghost" particle information for calculating molecular forces, as nodes can no longer rely on an invariant and regular arrangement of neighboring subdomains. By setting the notional Hilbert SFC lattice spacing to be $\geq r_{cut}$, each node may generate an internal list of the remote nodes with whom communication is required by periodically mapping local particle coordinates into SFC cells, and detecting any SFC cells which require the sharing of particle information (see **Fig. 3**).

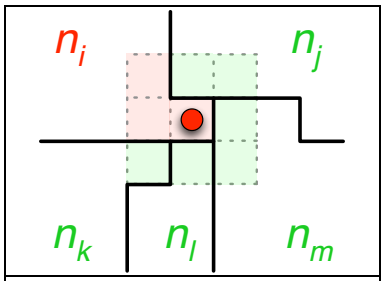

Figure 3: 2-D illustration of the dynamic communication mapping as described in the text. The notional 3x3 set of Hilbert SFC cells (highlighted) that surround a real particle in the local domain of node n_i are examined, and some cells are found to lie in the domains of nodes $n_{j\text{-}m}$. Information regarding this particle will therefore be shared with those nodes. Node domain boundaries are solid lines, SFC cells dashed lines.

It is important to note that although the approach in **Fig. 3** allows a node n_i to detect the need to share information with a node n_j, n_j does not automatically know to listen to n_i for incoming data. The use of MPI "collective" communications to share such information was found to be inefficient for large node counts N_{nodes}, and instead remote memory access (RMA) via the DMAPP interface of Cray's *Gemini* network hardware was used. Each node exposes a local memory array of N_{nodes} integers for RMA operations, and where n_i detects the need to share data with n_j then n_i writes a value of 1 directly into element i of n_j's exposed memory array. It is then straightforward for nodes to detect emerging communication patterns by examining their local memory arrays and detecting the nonzero elements that indicate the need to communicate with a particular remote node.

The dynamic communications mapping approach leads naturally to efficient single-pass sharing of particle data with asynchronous point-to-point routines. This is in contrast to algorithms that may require several communications "sweeps" on

each Cartesian axis [5,7]. The effects of communication latency can be reduced by maintaining two sets of Verlet lists on each node: the first set contains only interacting pairs of local particles, and the second contains interacting pairs which feature ghost particles. Nonbonded interactions between local real particles (which require no remote data to be received) can then be evaluated while waiting for ghost particle information to arrive from any remote data sources.

Inter-node communication in the CG-MD code is thus decoupled from any particular spatial arrangement of subdomains, providing automatic and highly dynamic runtime detection of emerging communication patterns. It is therefore possible, for example, to insert a completely new local molecule into the simulation and have this molecule automatically be split and distributed across the appropriate compute nodes even when the molecule spans multiple subdomains with no shared borders.

2. Initial performance testing

The test simulation consists of a model CG lipid bilayer membrane (BLM) of dimensions 1 μm x 1 μm, accompanied by two spherical BLMs of diameter 125 nm to provide a total system size of ~15.6 x 10^6 CG particles. (**Fig. 4**). Systems of this type have direct biological relevance in the context of e.g. the HIV viral lifecycle or chemical synapses[11]. The test simulation was executed on a range of 128 to 8192 compute nodes on the Blue Waters facility using 1, 4, 8 and 16 MPI ranks per compute node. Although Verlet lists and nonbonded interactions (via the Lennard-Jones 12-6 potential) were calculated in these simulations, the resultant nonbonded forces were not used in the integration of the equations of motion: this approach allowed *identical* trajectories to be generated both in the presence and absence of Verlet list generation and nonbonded force calculations, providing a direct measurement of the true influence on runtime performance of the communications routines and the nonbonded force calculations (with the latter typically the most time consuming part of an MD simulation).

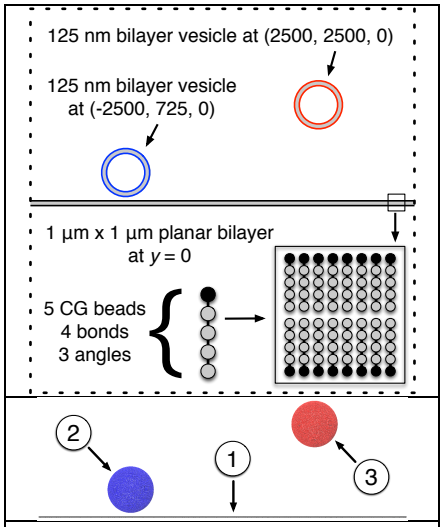

Figure 4: Illustration of the system as described in the main text. **Upper panel**: two spherical bilayer membranes (BLMs) are positioned above a 1 μm x 1 μm planar BLM. **Lower panel**: VMD snapshot of the region containing the planar BLM (1) and the two spherical BLMs (2 and 3). Area per lipid is 70 Å^2 for all BLMs. Periodic simulation cell is 1 μm x 1 μm x 1 μm, with r_{cut} = 2.0 nm.

Fig. 5 presents CG-MD code performance data as a function of CPU core count. Impressive "strong" scaling behavior is observed (**Fig. 5 a**), with scaling performance dropping below 85% of ideal only at ~131,000 CPU cores (where the CG-MD code is executing MD timesteps every ~1.6 milliseconds). One interesting aspect of **Fig. 5 a** is the presence of two effectively separate curves; the first curve features assignments of one, two and four MPI processes per compute node with the second curve containing eight and 16 MPI processes per compute node. This separation reflects the Blue Waters hardware,

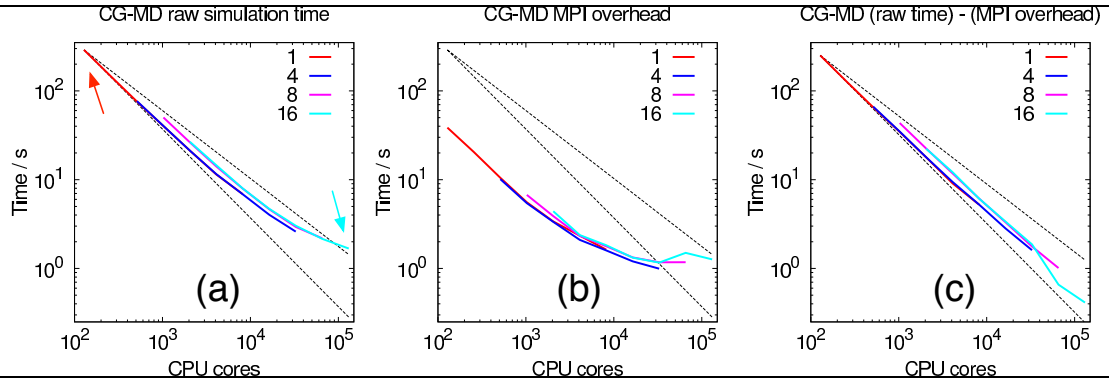

Figure 5: Raw simulation time required for 1000 MD steps of the test simulation as a function of CPU core count (**a**), the times for identical simulations in the absence of Verlet list generation and nonbonded force calculations (**b**), and the difference between (**a**) and (**b**) is shown in (**c**). Line color indicates MPI ranks per compute node, with ideal scaling (lower dashed line) and 85% of ideal scaling (upper dashed line) shown. Raw time for 128 CPU cores is indicated by the red arrow, 131,072 CPU cores with a blue arrow.

as more than four MPI ranks per compute node requires the sharing of L3 cache memory with noticeable performance effects. **Fig. 5 b** depicts the time required for identical simulations in the absence of Verlet list generation and nonbonded force calculations and therefore indicates the general MPI communications overhead (bond and angle forces contribute < 5% of these times). From this figure we observe that the limiting factor for the scaling of this system is actually MPI communications overhead. This overhead is mainly asynchronous point-to-point communications rather than MPI collective communications. **Fig. 5 c** shows the raw simulation times with the MPI overhead subtracted, with the Hilbert SFC shown to provide excellent load balancing performance over at least three decades of logarithmic CPU core counts, even for such an extremely heterogeneous CG system. Note that this testing involves a relatively small system using a computationally inexpensive Lennard-Jones 12-6 pair potential, and the CG-MD code can use much larger CPU core counts if given larger/denser systems or more expensive interaction potentials.

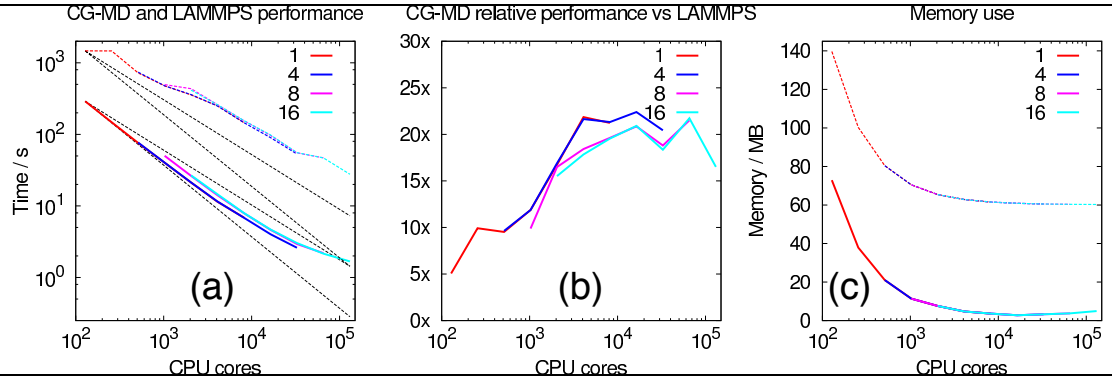

Figure 6: Performance of the CG-MD code (solid lines) and LAMMPS (dashed lines) for the same system and CPU core arrangements, with ideal and 85% of ideal scaling shown for both data sets (**a**); CG-MD performance relative to LAMMPS (**b**); CG-MD memory usage compared to that of LAMMPS (**c**).

Fig. 6 compares the performance of the CG-MD code to LAMMPS for the test simulations. In the absence of load balancing, LAMMPS is unable to achieve 85% scaling efficiency for any CPU core count above the initial 128 cores (**Fig. 6 a**);

although a load balancer has recently been added to LAMMPS, this load balancer halted on simulation startup with an error message for this system. The CG-MD code becomes progressively faster compared to LAMMPS as the CPU core count is increased (**Fig. 6 b**) until MPI communication overheads begin to hinder this increase in relative performance for the largest core counts. In all cases, CG-MD code memory use is significantly less than that reported by LAMMPS (**Fig. 6 c**).

3. Conclusions and future work

The CG-MD code displays excellent performance for CG systems with extremely heterogeneous particle distributions, limited only by the overheads of MPI's point-to-point communication routines for very high speeds at large CPU core counts (~1.6 ms per MD timestep at 130,000+ cores). Alongside the sparse memory characteristics and dynamic topological calculations, the CG-MD code therefore offers a platform for entirely new classes of dynamic cell-scale molecular simulations using CG models with implicit solvent. The Hilbert SFC load balancer concentrates the MD calculations very effectively over the available compute resources, and should therefore perform well in combination with GPU accelerated calculations. One potential avenue of investigation to reduce the MPI overhead is to replace the appropriate MPI point-to-point routines with calls to entirely hardware-native APIs such as DMAPP, at the cost of some additional complexity and portability considerations in the CG-MD software.

References

1 Allen, M. P. & Tildesley, D. J. *Computer simulation of liquids*. (Clarendon, 1987).

2 Frenkel, D. & Smit, B. *Understanding molecular simulation : from algorithms to applications*. 2nd ed. edn, (Academic, 2002).

3 Leach, A. R. *Molecular modelling : principles and applications*. 2nd ed. edn, (Prentice Hall, 2001).

4 Rapaport, D. C. *The art of molecular dynamics simulation*. 2nd ed. edn, (Cambridge University Press, 2004).

5 Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular-Dynamics. *J Comput Phys* **117**, 1-19, doi:Doi 10.1006/Jcph.1995.1039 (1995).

6 Verlet, L. Computer "Experiments" on Classical Fluids. 1. Thermodynamical Properties of Lennard-Jones Molecules. *Phys Rev* **159**, 98-103, doi:10.1103/PhysRev.159.98 (1967).

7 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* **4**, 435-447, doi:Doi 10.1021/Ct700301q (2008).

8 Phillips, J. C. *et al.* Scalable molecular dynamics with NAMD. *J Comput Chem* **26**, 1781-1802, doi:Doi 10.1002/Jcc.20289 (2005).

9 Lawder, J. *The application of space-filling curves to the storage and retrieval of multi-dimensional data*, 1999).

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

11 Hu, Y. M., Qu, L. & Schikorski, T. Mean Synaptic Vesicle Size Varies Among Individual Excitatory Hippocampal Synapses. *Synapse* **62**, 953-957, doi:Doi 10.1002/Syn.20567 (2008).