University of Colorado REU in Computational Condensed Matter Physics

C r' F'n

A g t , 2

Introduction

My work in the 2006 REU program at the University of Colorado involved molecular dynamics simulation of condensed matter systems under Professor Matthew Glaser and graduate student Zachary Smith. The first half of the summer was spent writing computer code simulating mixtures of spheres and spherocylinders. The second half of the summer this code was used to measure the diffusion properties of a carbon nanotube in an aqueous environment. In this paper I describe two basic computational methods of molecular dynamics simulation I learned over the course of the summer in Section I, and in Section II I describe the results of the carbon nanotube simulation.

1 Computational Methods of MD Simulation

Molecular dynamics (MD) simulation generally involves computationally time-evolving molecular systems and calculating time averages of desired thermodynamic properties. Such an approach may be contrasted with molecular Monte Carlo simulations, which sample the phase space of systems (independent of time) to calculate thermodynamic properties. The most computationally expensive portion of virtually any MD simulation is the calculation of inter-particle forces, so much effort is expended in developing efficient particle interaction routines. There exist two standard approaches which greatly improve the "all pairs" force calculation algorithm.

The all pairs algorithm is the simplest method of calculating inter-particle interactions. To determine the net force exerted upon a given particle, it sums the forces exerted upon the given particle by *all other particles* in the system. In a system of N particles, this requires N^2 forces calculations. Invoking Newton's Third Law improves the situation slightly, requiring $N(N-1)/2$ force calculations, but the algorithm still scales as N^2. Another common practice for short-range systems is to institute a maximum distance, r_cutoff, which forms a "halo" around a given particle. Only the forces contributed by particles within the given particle's halo are summed to give the total force on the particle–the rest of the forces are discounted. This makes sense for short-range systems because beyond r_cutoff (which is usually equal to 2.5σ, where σ is the characteristic length of the potential function) the contributing forces become vanishingly small.

Yet even this trick does not provide a substantial boost in efficiency, for the program must still check the separation distance of every particle pair combination in order to filter those separated by more than r_cutoff. For this reason, such a routine is still considered an N^2, "all pairs" algorithm. To mitigate this poor efficiency, MD simulations typically employ either of two tricks: the neighbor list or the cell list. Both are straightforward concepts, but they require a fair bit more programming than the naive all pairs approach.

The neighbor list scheme begins in the same way as the all pairs scheme, by placing a halo of radius r_cutoff around a given particle. Neighbor list

Figure 1: Neighbor list cartoon.

schemes then go one step further and place another halo (also centered around the particle) a certain distance beyond the first halo. This distance is referred to as the skin distance, and the space between the first and second halos is called the skin (see Fig. 1). Every particle in the system is assigned an identification number, and linked lists are then utilized to record which particles are within $r\_cutoff + skin\_distance$ (the second halo) of the given particle. To calculate the force on the given particle, the program needs not check *every* particle in the system for being within r_cutoff of the given particle, but only those particles on the linked list [1, pp. 147-149]. Of course, after some time new particles may enter or exit the region of space encompassed by the second halo, and the list must be updated with sufficient frequency such that no particle comes within r_cutoff of the given particle without being detected. This is achieved by keeping track of the displacements of all particles in the system since the most recent neighbor list update. Whenever any particle's displacement magnitude exceeds $skin\_distance/2$, the neighbor lists must be updated. (The prescribed update distance is $skin\_distance/2$, and not $skin\_distance$, because in the worst-case scenario one particle may be on the verge of another particle's outer halo. If they both move directly toward each other, they will be separated by just a shade over r_cutoff after both have moved $skin\_distance/2$.)

It is advantageous to use neighbor lists only when the number of particles within each outer halo (n_{h}) is much less than the total number of particles within the system. The number of particles within each outer halo is (on average) given by $n_{\mathrm{h}} = \frac{4}{3}\pi r_{\mathrm{h}}^3 \rho$, where r_{h} is the radius of the outer halo (typically 2.7σ) and ρ is the particle density [2, p. 552].

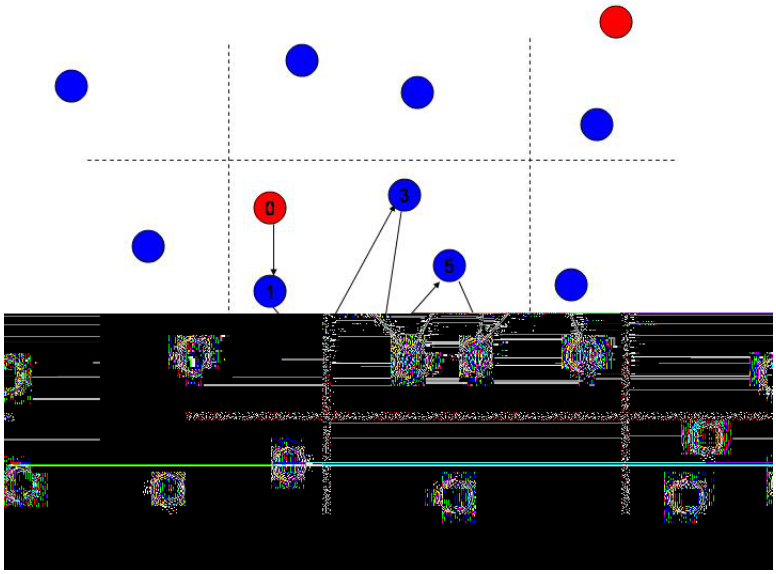

Figure 2: Cell list cartoon.

To compare neighbor list efficiency to all pairs efficiency, first realize that because all pairs requires $N(N-1)/2$ calculations to compute the total energy of the system, the computational time is simply $\tau_{\text{allpairs}} = cN(N-1)/2$, where c is the computational time per particle pair. Neglecting updates, the computational time required using the neighbor list scheme is $cn_{\text{h}}N$, which is obviously a marked improvement if $n_{\text{h}} \ll N$. Taking updates into account, however, introduces the extra term $\frac{c_{\text{h}}}{n_{\text{u}}}N^2$, where c_{h} is the computational time taken to determine if a particle pair is separated by less than $r\_cutoff + skin\_distance$, and n_{u} is the average number of iterations after which the neighbor lists must be updated. The overall efficiency of the neighbor list scheme is therefore

$$\tau_{\text{nl}} = cn_{\text{h}}N + \frac{c_{\text{h}}}{n_{\text{u}}}N^2. \tag{1}$$

"Experimentation" must be performed to determine what skin length gives the most efficient combination of n_{h} and n_{u}.

In the cell list scheme, the system unit cell is partitioned into a grid, with each unit constituting a cell. Two arrays are maintained–one which designates the "head" particle in each cell, and another which lists all other particles in each cell (see Fig. 2). To check for particles within r_cutoff of a given particle, the program simply checks the surrounding cells rather than the entire system. If the cells' dimension of minimum length measures r_cutoff or greater, then the program needs only check one cell deep in all directions. In contrast to neighbor lists, cell lists must be updated every iteration. Fortunately, this takes

only N calculations. The efficiency of the cell list scheme is given by

$$\tau_{\mathrm{cl}} = cn_{\mathrm{c}}N + c_{\mathrm{cl}}N, \tag{2}$$

where $n_{\mathrm{c}} = 27\rho r_{\mathrm{c}}^3$ and c_{cl} is the computational time taken to determine the cell occupied by a given particle [2, p. 553].

The most efficient MD routines usually employ a combination of both the neighbor list and cell list approaches. Neighbor lists are used to directly calculate inter-particle forces, and cell lists are implemented to speed neighbor list updates. Note that with this approach cell lists need not be updated every iteration, but only when the neighbor lists need updating. The efficiency of this algorithm is then [2, p. 553]

$$\tau_{\mathrm{nlcl}} = cn_{\mathrm{h}}N + \frac{c_{\mathrm{cl}}}{n_{\mathrm{u}}}N. \tag{3}$$

2 Diffusion Properties of Carbon Nanotube in Water

In his mathematical elucidation of Brownian motion in 1905, Einstein derived the following Gaussian probability distribution for a one-dimensional Brownian particle [3, p. 461]:

$$p(t,x) = \frac{1}{\sqrt{4\pi Dt}} \exp\left[-\frac{(x-x_o)^2}{4Dt}\right]. \tag{4}$$

From this equation it is clear that the constant D determines the rate at which the probability distribution of the particle spreads, or *diffuses*, over time. This diffusion constant is a property of the size and shape of the particle, as well as its interactions with surrounding particles. In addition to changing position over time, spherically asymmetric particles also change in orientation. The rotational diffusion constant is thus a measure of the rate of such a particle's stochastic change in orientation.

Many molecules of interest, including proteins, nucleic acids, viruses, and carbon nanotubes, exhibit approximately stiff rod-like conformations. Recent work has modeled short double-stranded DNA segments (with lengths of less than 50 nm) as rod-shaped particles [4], and the diffusion properties of carbon nanotubes have recently been explored through MD simulation [5]. To better understand the properties of such molecules, we devised a two-dimensional MD simulation to measure the rotational and translational diffusion coefficients of rod-shaped molecules in water. As a case study, we simulated a carbon nanotube (with a mass of 2700 amu, a length of 10 nm, and a diameter of 1 nm) in an explicit solvent of water molecules held at a constant temperature of 298 K. The system was held at constant volume, with a pressure of approximately 0.065 J/m^2. Particles interacted via a Lennard-Jones potential, given by

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]. \tag{5}$$

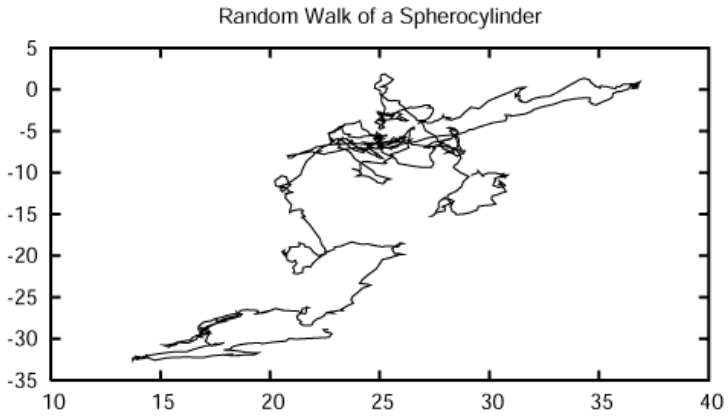

Figure 3: Random walk of carbon nanotube in one arbitrary simulation run.

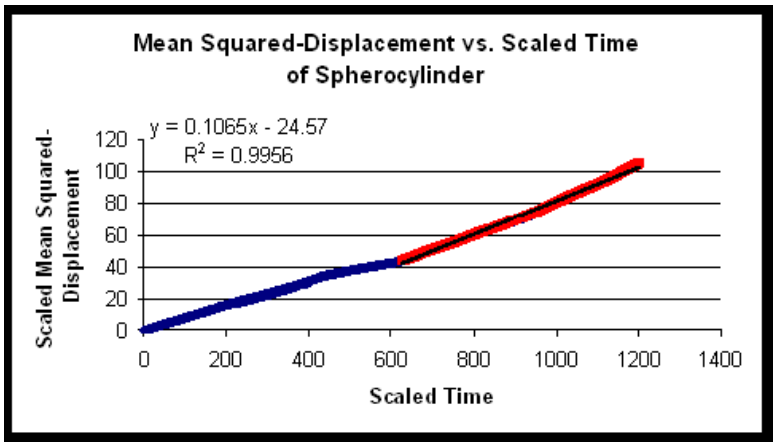

Figure 4: $< x^2 >$ vs. machine time for carbon nanotube.

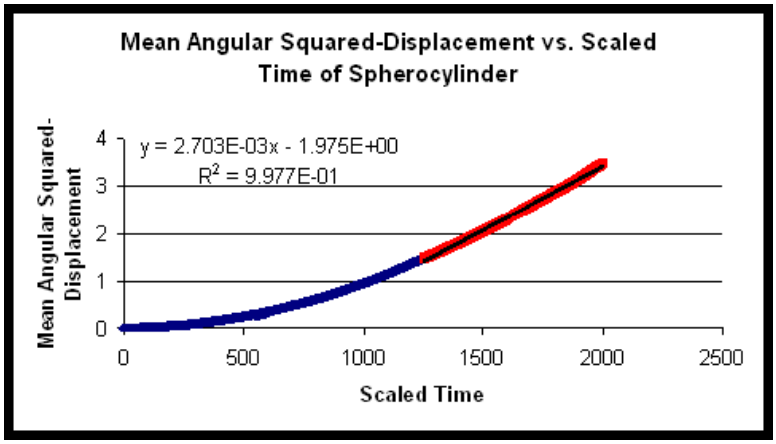

Figure 5: $< \theta^2 >$ vs. machine time for carbon nanotube.

Here ϵ sets the energy scale of the system, and σ sets the length scale. These parameters were chosen to match those in [6], so $\epsilon = 0.79$ kcal/mol and $\sigma =$ 0.29nm.

We ran several simulation runs (see Fig. 3 for the random walk taken by the spherocylinder in one simulation) and then plotted the mean-squared displacement and the mean-squared angular displacement of the spherocylinder versus the scaled machine time. Figures 4 and 5 give the slopes of these data sets. Note that we fit only the linear portions of the data; at short times, a particle in Brownian motion maintains a "memory" of its initial position or orientation, and only after this short-term memory has been lost does the particle enter the linear regime.

To calculate the two diffusion constants from the slopes of these graphs, we must first convert the slope values from machine units to SI units, then determine how the diffusion coefficient is mathematically related to the mean-squared displacement. The translational diffusion coefficient is given in units of length-squared per unit time; the unit of length of the system was set by the diameter of a water molecule, so one machine unit of length corresponded to 0.29 nm. Our simulation sampled the data once every five units of machine time, each of which were 7×10^{-10} seconds in length [6, p. 3101], so the slope from Fig. 4 converts to $m_{\text{trans}} = 2.56 \times 10^{-8}$ cm^2/s. Likewise, the slope from Fig. 5 converts to $m_{\text{rot}} = 7.72 \times 10^5$ rad^2/s.

To ascertain the mathematical relationship between the particle's mean-squared displacement and its diffusion coefficient, we start by determining the dependence of the mean-squared displacement expectation value of a one-dimensional Brownian particle upon time, which is given by the equation (for $x_o = 0$ at $t_o = 0$)

$$< x^2 >= \frac{1}{\sqrt{4\pi Dt}} \int_{-\infty}^{+\infty} x^2 \exp\left[-\frac{(x)^2}{4Dt}\right] dx. \tag{6}$$

The integral $\int_{-\infty}^{+\infty} x^2 \exp\left[-ax^2\right] dx$ is equal to $\sqrt{\pi}$, so we obtain the simple result

$$< x^2 >= 2Dt. \tag{7}$$

The preceding derivation was for assumes a one-dimensional system; the added degrees of freedom in higher dimensions result in the modification

$$< x^2 >= 2dDt, \tag{8}$$

where d is the dimensionality of the system. Relating this result to the slopes of Figures 4 and 5 (and realizing that the system is two-dimensional) yields the equations

$$D_{\text{trans}} = \frac{m_{\text{trans}}}{4} \tag{9}$$

and

$$D_{\text{rot}} = \frac{m_{\text{rot}}}{4}. \tag{10}$$

The translational diffusion coefficient of the carbon nanotube is therefore $D_{\text{trans}} = 6.40 \times 10^{-9}$ cm^2/s , and the rotational diffusion constant is $D_{\text{rot}} = 1.93 \times 10^5$ rad^2/s.

3 Conclusion

The preceding calculations of carbon nanotube diffusion coefficients must be considered rough and preliminary, for no statistical analysis has yet been conducted. The translational diffusion coefficient is several orders of magnitude below that found experimentally by Pecora [4, p. 21] for a similarly-sized segment of double-stranded DNA, though our calculation is expected to yield a much smaller value since our system was two-dimensional system. Whether or not our simulation agrees with experiment in three dimensions has not yet been determined. Additional work is clearly required to obtain improved, more reliable results.

My research this summer was an incredible experience. I learned an incredible amount about MD simulation and condensed matter physics, and I wish to thank Zach Smith and Matt Glaser for their patience and eagerness to answer any and all of my questions.

References

[1] M.P. Allen & D.J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, NY, 1987.

[2] D. Frenkel & B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, London, England, 2nd edition, 2002.

[3] R.K. Pathria. *Statistical Mechanics*. Elsevier, New Delhi, India, 2nd edition, 2004.

[4] R. Pecora. *Macromol. Symp.* **2005**, 229, 18-23.

[5] S. Jackobtorweihen, F.J. Keil, & B. Smit. *Temperature and Size Effect on Diffusion in Carbon Nanotubes* (*J. Phys. Chem. B*).

[6] Y. Lansac, Prabal K. Maiti, M. Glaser. *Coarse-grained simulation of polymer translocation through an articial nanopore* (*Polymer*). **2004**, 45, 3099-3110.