CHRONOGRATOR

A Multiple Time Step Integrator

Jaro Camphuijsen (6042473) and Rahiel Kasim (10447539)

22 February 2016

Contents

Abstract

There are numerous computational methods to speed up molecular dynamics simulations. In this paper we will show how to combine a multiple time step integrator with Verlet lists by implementing these schemes into an existing molecular dynamics simulation. In simulations of large systems we often use a cutoff radius to limit the interaction pairs of a particle within a computationally favourable distance. The Verlet list is a bookkeeping method to keep a list of the particles within the cutoff radius, so we lose the N^2 dependency for every integration step. The multiple time step integrator allows using a smaller time step for particles that are closer to each other and thus contribute more to the particle's acceleration due to the shape of the Lennard-Jones potential, while particle pairs on larger distances, where the potential varies less, are integrated with a bigger time step, saving CPU cycles.

1 Introduction

Molecular dynamics (MD) is a computational technique used to study the properties of systems of atoms and molecules, for example of fluids. Such a simulation involves integrating the system according to Newton's laws of acceleration due to the intermolecular forces until it reaches equilibrium, and then to measure the property we're interested in. The interaction between particles is often modeled using a Lennard-Jones potential (L-J potential). This is an approximation of the effective potential between pairs of (neutral) atoms or molecules. [1]

The L-J potential for a particle at the origin is shown in figure 1. The distance on the x-axis is in units of σ (effective radius of the particle), and the energy on the y-axis is in units of ϵ, the (absolute) energy at r_m in the potential. We see that the potential is steep at short distances, and doesn't change much at the long range. The integration time step of a MD simulation should always be smaller than the shortest relevant time scale in the simulation. [2] That is why it is interesting to look at multiple time step integrators: you could integrate particle pairs at short distances with a smaller time step than the particles on a bigger distance. This bigger time step will save computational time while the shorter time step increases the accuracy of the simulation.

Martyna et al. [3] have reported a time-reversible multiple time step algorithm for *NVE* simulations. Time-reversible schemes are also symplectic schemes: the Hamiltonian is a conserved quantity for the scheme. [4] It is important to use schemes with this property in MD because they conserve the energy in the system. In schemes lacking time reversibility, like Runge-Kutta and the Euler method, the energy in the system increases due to integration errors.

The particles at short and long distances in the L-J potential can be distinguished using Verlet lists. This is a clever bookkeeping technique used to prevent needing to recalculate if a particle pair has to be integrated with the small or the big time step. In the following section we will look at the practical combination of a time-reversible multiple time step integrator with Verlet lists.

2 Methods

To implement a multiple time step integrator (MTS) we need a way to split up the computation of forces into a short range and long range part. We split up the L-J potential into the short range repulsive part and the long range attractive part. We use a double Verlet list algorithm to hold track of particles at close or long range. By implementing a single Verlet list we can already speed up the simulation because we ignore particles that are outside the cutoff radius. By implementing a second Verlet list for each particle i with its own (smaller) cutoff radius, we can hold track of the particles that are close range neighbours and exert large close range forces on i. These close range forces then get evaluated n times with a timestep dt every major time step Δt, so $dt = \Delta t/n$.

2.1 Verlet List

Evaluation of the interaction potential goes over all particles in the system, however when we use a truncated L-J potential, most of the particles in the system do not contribute to the force because the potential is flat and zero. It

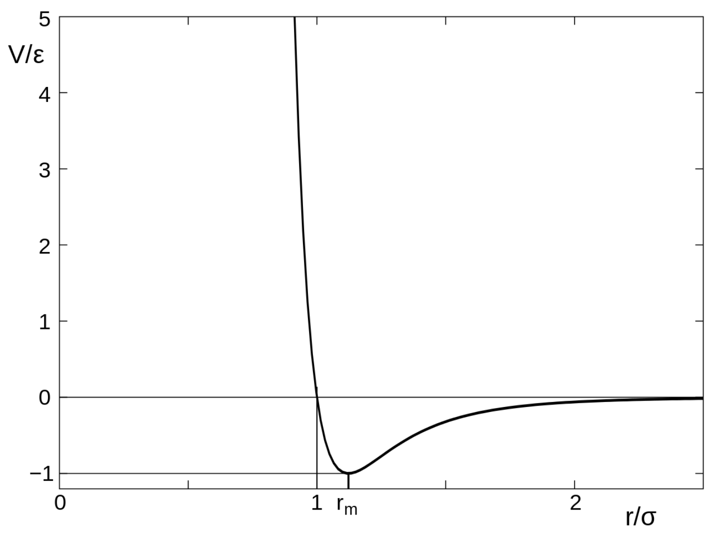

Figure 1: The Lennard-Jones effective potential between two particles. The x-axis shows the distance from the particle at the origin and the other particle in units of the radius (σ) of the origin particle. The y-axis shows the potential in units of ϵ, the (absolute) energy at r_m in the potential.

would be advantageous to only take into account the particles inside the cutoff radius of the particle in question. This implies we need to maintain a list of neighbours for each particle in the system and update this list every once in a while. In the Verlet method we define a second radius r_v larger than r_c and check at every force calculation whether the particle has moved more than the difference between those radii since the last time the list was updated. This check is shown in the following code snippet.

```
if (abs(x(i) - xv(i)).gt.rdver) then
    call vlist()
end if
```

Here `x(i)` is the current position of particle `i` and `xv(i)` is the position of `i` at the last time its Verlet list was updated. `rdver` is the difference between r_v and r_c. Updating the Verlet list includes looping over all particles and adding the particles which are within r_v to the proper Verlet list. Therefore we need to make sure the Verlet list is updated far less than every timestep, which means that `rdver` must be significantly greater than the mean position change per time step. Otherwise no CPU time benefit is obtained. On the other hand our Verlet list should not be so large as to include all particles in the system but only a small fraction. Also when dealing with a system with periodic boundary conditions, we need to make sure our Verlet radius is smaller than the periodic box. When looping over all particles in the system we then take the nearest mirror image of the selected particle. Below the Verlet list update scheme is given in pseudocode for one dimension (from Appendix F [2]).

```
SUBROUTINE vlist

do i=1, npart
    nlist(i) = 0
    xv(i) = x(i)
end do
do i=1, npart-1
    do j=i+1, npart
        xr = x(i) - x(j)
!   Get nearest mirror particle
        if (xr.gt.hbox) then
            xr = xr - box
        else if (abs(xr).lt.rv) then
            xr = xr + box
        endif
!   Make new Verlet list
        if (abs(xr).lt.rv) then
            nlist(i) = nlist(i) + 1
            nlist(j) = nlist(j) + 1
            list(i, nlist(j)) = j
            list(j, nlist(j)) = i
        end if
    end do
end do
return
end
```

The original force calculation would loop over every particle in the system and for every particle calculate the interaction with every other particle in the system. This is an algorithm of order N^2. The force calculation in the Verlet list simulation only has to loop over all particles once and for each particles calculate only the interaction with the particles in its Verlet list. The order of this scheme depends on the size of r_v but is in general much less than N^2. To find a proper value of r_v we can minimize the CPU time with variable r_v.

2.2 Multiple Time Step Integrator

For the MTS we need to split the potential into two parts, a close range potential, which in case of the L-J potential is the steep repulsive part, and a long range potential which will consist of the attractive part up to the cutoff radius. We can take a proper close range cutoff radius r_{c2} by minimizing the potential (using reduced units where $\epsilon = \sigma = 1$):

$$\frac{d\mathcal{U}_{LJ}}{dr} = 4\left[12\left(\frac{1}{r}\right)^{11} + 6\left(\frac{1}{r}\right)^{5}\right] = 0 \qquad (1)$$
$$r_m = 2^{1/6} \approx 1.122$$

The attractive and repulsive force annihilate each other at this point. For smaller r the steep repulsive potential results in a stronger force and accelerations on short time scales while between r_m and r_c we have the smaller and more steady attractive potential which will influence particles on larger timescales. For this long range force more particle interactions have to be calculated since the number of particles in a radial shell scales with r^2. In the following pseudo code the MTS subroutine is shown for one dimension (equation 58 in [3]). The subroutine `force(i, rc)` calculates the force on particle `i` up to the specified cutoff radius `rc` which can either be r_c for the total force calculation or r_{c2} for calculating the short range forces.

```
SUBROUTINE multi

do i=1, npart
    vx(i) = vx(i) + 0.5 * Delt * (Fx(i) - (Fx2(i))
enddo
do j=1, n
    do i=1, npart
        vx(i) = vx(i) + 0.5 * dt * Fx2(i)
        x(i) = x(i) + 2 * dt * vx(i)
        call force(i, rc2)
        vx(i) = vx(i) + 0.5 * dt *Fx2(i)
    enddo
enddo
do i = 1, npart
    call force(i, rc)
    vx(i) = vx(i) + 0.5 * Delt * (Fx(i) - Fx2(i))
enddo
return
end
```

To make the MTS feasible we maintain two different Verlet lists, one with a cutoff radius at the original r_c which will be used to calculate the total force on particles at each major time step Δt and an additional list with a cutoff radius at r_{c2} to calculate only the short range forces at every minor time step dt. In the force subroutine we had implemented the Verlet list check and this can be extended to the multiple time step algorithm if we make sure to maintain two different Verlet list with their appropriate cutoff radii.

By separating the short-range and long-range particles we also effectively split the L-J potential in two: there is a discontinuity at the boundary between short and long. (The short-range potential is abruptly zero for $r > r_{c2}$ and the long-range potential is zero for $r < r_{c2}$.) To prevent discontinuities we make use of a switching function $S(r)$ that uses an interpolation scheme to smoothly escort the potential to zero. We use the switching function as follows [3]:

$$\begin{aligned} f_{tot}(r) &= f_{\text{short}}(r) + f_{\text{long}}(r) \\ f_{\text{short}}(r) &= S(r) \times f(r) \\ f_{\text{long}}(r) &= [1 - S(r)] \times f(r), \end{aligned}$$

where

$$S(r) = \begin{cases} 1 & 0 < r \leq r_{c2} - \lambda \\ 1 + \gamma^2(2\gamma - 3) & r_{c2} - \lambda < r < r_{c2} \\ 0 & r_{c2} \leq r \leq r_c \end{cases}$$

and

$$\gamma = \frac{r - r_{c2} + \lambda}{\lambda}.$$

We've implemented this switching function in the calculation of the short range forces from the second Verlet list.

To asses the accuracy of an integration scheme, and to fairly compare them, we can compute the average energy drift, as defined by Martyna et al. [3]:

$$E = \frac{1}{N_{\text{step}}} \sum_{i=1}^{N_{\text{step}}} \left| \frac{E(i\Delta t) - E(0)}{E(0)} \right|, \tag{2}$$

where $E(0)$ is the first energy measured (after equilibration) and $E(t)$ the total energy at time t.

Finally we define the efficiency of a scheme as [2]:

$$\eta \equiv \frac{\Delta t \cdot n_{\text{steps}}}{t_{\text{CPU}}}, \tag{3}$$

which is just the length of the simulation divided by the CPU time. This efficiency is a relative measure of CPU time which we use to compare simulations with different simulation lengths.

3 Results

We started with an existing *NVE* simulation code [5] that used the Leapfrog integration scheme. This code was sped up by implementing a Verlet neighbor list with radius $r_v > r_c$. The main benefit of the Verlet List is in the calculation of the force between particles: instead of checking for all the particles if they are within the cutoff of the potential, we only have to check the particles in the Verlet list.

The magnitude of the Verlet radius affects the total CPU time of the program, as illustrated by Table 5.1 from ref. [6]. In figure 2 we show the total CPU time of the simulation over the magnitude of the Verlet radius. The first time at $r_v = 2.5$ is the time with no Verlet list. In this case we again loop over all particles every time step since the Verlet list check is always true. We see that there is an optimal value for r_v. The time increases after this value because for bigger Verlet radii there are more particles in the Verlet list and the program has to check for more particles if they are within the cutoff radius or not.

Our second result concerns the MTS with two Verlet lists. First we compare the energy drift of the MTS and the Leapfrog integrator. In figure 3 we see the energy drift as calculated by eq. 2 versus the stepsize. It can be seen that the MTS integrator has a consistently smaller energy drift.

The MTS simulation was tested for different values of Δt (time step length) and n (number of minor time steps in MTS). Here we look at the efficiency of

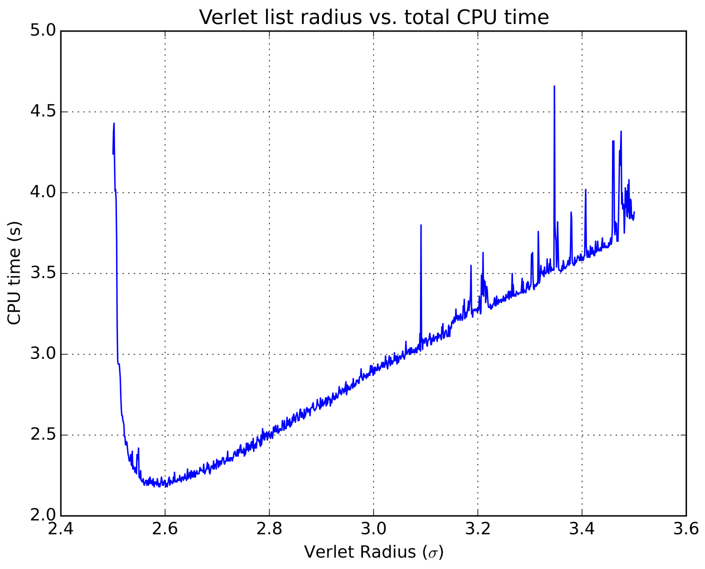

Figure 2: The CPU simulation time over varying Verlet radii used in the Verlet List method. The minimum CPU time was found at $r_v = 2.579$. The time was measured by using the `time` program where the total CPU time is the "user" plus "system" times. The radius of $r_v = r_c = 2.5$ is equivalent to having no Verlet List.

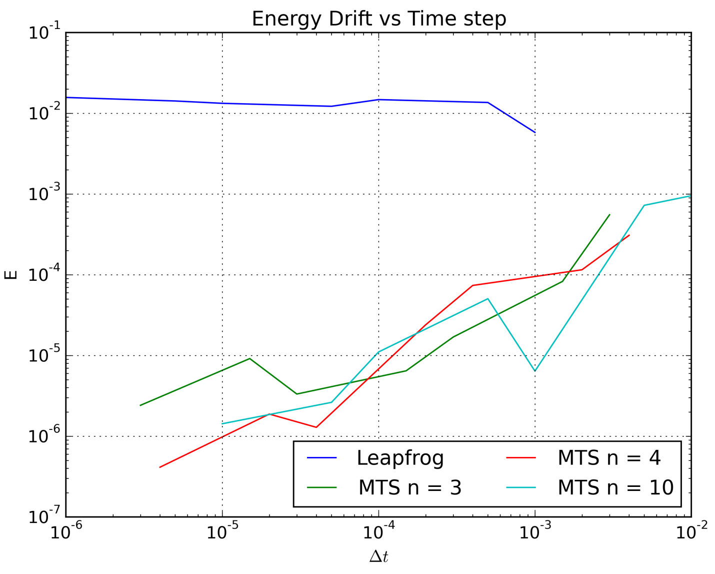

Figure 3: The energy drift versus the stepsize for the Leapfrog integrator and several MTS schemes with varying number of minor steps. The MTS outperforms the Leapfrog in terms of energy drift.

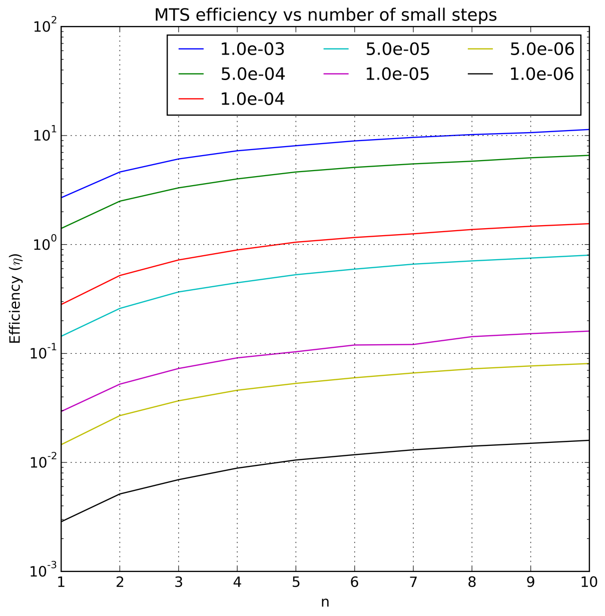

Figure 4: The efficiency vs. the number of small steps in the MTS. Shown in the legend is the small step size, δt in $\Delta t = n\delta t$.

the scheme for different numbers of small time steps. In figure 4 the efficiency of the simulation is plotted against the number of minor time steps per cycle. We can see that for all stepsizes a higher value of n leads to a higher efficiency. Another observation is that a bigger stepsize entails a higher efficiency. This can easily be understood: the simulation time is the same while cpu time has decreased because it takes a lower number of integration steps.

Finally we compare the efficiency of the Leapfrog integrator with Verlet list (using the optimal Verlet radius $r_v = 2.58$) and MTS integrators. In figure 5 we see that the Leapfrog has a higher efficiency than the MTS integrator with $n = 1$. However for all $n > 1$, the MTS integrator has a higher efficiency.

These results can be reproduced by running the simulation with the provided run scripts at GitHub [7].

4 Conclusions

Molecular dynamics programs can be made faster with no loss of accuracy and relatively few changes by implementing a Verlet neighbour list. Implementing a multiple time step integrator and splitting the Lennard-Jones potential in short- and long-range parts requires more effort and thoughtful programming. As we can see from figure 5 the leapfrog algorithm has a higher efficiency than MTS with $n = 1$. In this case we actually have only one time step and the algorithms should be equivalent. The lower efficiency of the MTS is due to the generation of the extra Verlet list for the short range forces. However the results show

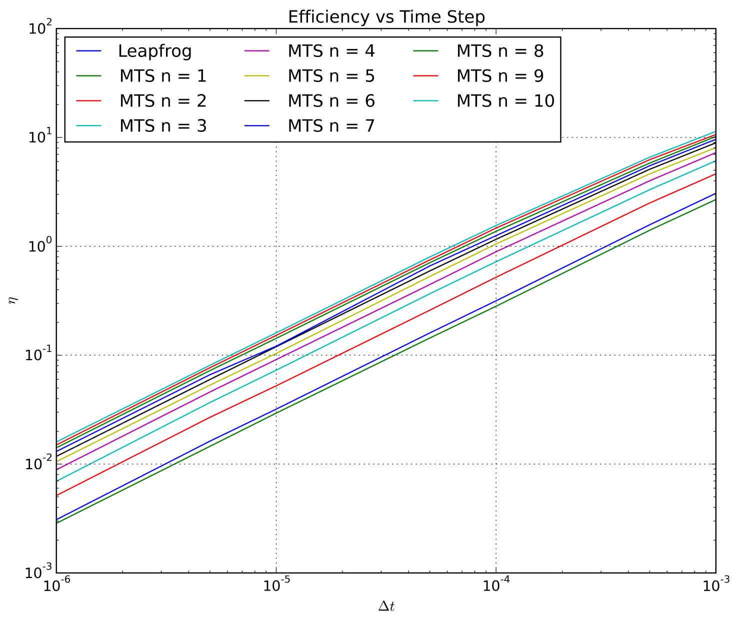

Figure 5: The efficiency vs. step size for the Leapfrog and MTS integrators. The number of small time steps n is shown in the legend.

that by increasing n we immediately gain higher efficiency and a smaller energy drift. The efficiency also increases linearly with increasing Δt but as we would expect, the accuracy goes down as well. This trade-off can be seen in figure 3 where the energy drift increases for increasing Δt. Overall the MTS simulation has a lower energy drift than the original leapfrog algorithm, and it is therefor a useful improvement on the previous algorithm.

5 Recommendations

Instead of a classic Verlet list it would be interesting to implement a cell list algorithm. In this algorithm the system is divided in cells of size $r_c \times r_c$ and for each cell a list is maintained. Such a list updates once a particle enters or leaves the cell and the update subroutine does not loop over all particles since it is known which particle left or entered which cell. The cell list can be an improvement on computational time, especially for large numbers of simulation particles as can be seen in Figure F.4 from ref. [2].

References

[1] Wikipedia. Lennard-Jones potential — Wikipedia, The Free Encyclopedia, 2015. [https://en.wikipedia.org/w/index.php?title=Lennard-Jones_potential&oldid=694525432; accessed 27-January-2016].

[2] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation: From Algorithms to Applications.* Academic Press, second edition, 2002.

[3] Glenn J. Martyna, Mark E. Tuckerman, Douglas J. Tobias, and Michael L. Klein. Explicit reversible integrators for extended systems dynamics. *Molecular Physics*, 87(5):1117–1157, 1996.

[4] Wikipedia. Symplectic integrator — wikipedia, the free encyclopedia, 2016. [`https://en.wikipedia.org/w/index.php?title=Symplectic_integrator&oldid=704657907`; accessed 13-February-2016].

[5] Daan Frenkel & Berend Smit. Case Study 4: Static Properties of the Lennard-Jones Fluid. [`http://www.acmm.nl/molsim/frenkel_smit/CaseStudy_4/index.html`; accessed 16-January-2016].

[6] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids.* Clarendon Press, Oxford, 1987.

[7] Jaro Camphuijsen, Rahiel Kasim. Chronogratoor. [`https://github.com/sunsistemo/chronogrator`; accessed 22-Februari-2016].