Molecular dynamics Simulation using Verlet algorithm and linked cells

Computational Statistical Physics, Exercise 7 and 8
Sebastian Keller, sebkelle@ethz.ch

1 Introduction

In this exercise a C++ simulation program was written to simulate the Newtonian dynamics of point-like particles.

2 Theory

2.1 Verlet propagation

The Verlet scheme for propagating particles over discrete time steps reads:

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + \Delta^2 t\, \ddot{x}(t)$$

2.2 Force calculation

The force acting on every particle can be calculated as:

$$\ddot{x}_i = \frac{1}{m_i} \sum_j f_{ij}(t) \quad , \quad f_{ij} = -\nabla V(r_{ij}(t))$$

2.3 Lenard Jones potential

In this simulation the Lenard Jones potential V_{LJ} was used as the potential between the particles.

$$V_{\mathrm{LJ}} = 4\epsilon\Big[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}\Big]$$

2.4 Contact time calculation

The following formula describes the time it takes for two particles to enter and leave their interaction area ($2.5 \cdot \sigma$ for the Lenard Jones potential) in a 1D collision.

$$t_c = 2 \int_{r_{\min}}^{r_{\max}} \frac{1}{\sqrt{\frac{2}{\mu}(E - V(r))}}\, \mathrm{d}r,$$

where μ is the reduced mass of the particles colliding and E the maximum kinetic energy.

This value can be used to estimate the timestep needed for the simulation. In this simulation $\Delta t = t_c/40$ was used.

3 Implementation details

The simulation code was written in an object oriented style involving the following classes:

- space<P>, templated over the particle type, holds the particles (→ Exercise 7)
- lc_space<P>, linked cell version, as well templated over the particle type (→ Exercise 8)
- point_particle<numeric_t>, a point like particle, which knows about its mass and to Verlet-propagate itself
- various function objects, like the Lenard Jones potential and buffered functors.

3.1 Initialization

I've implemented either placing the particles initially in a 2D square grid (xy plane, for making videos) or in a 3D cubic grid. Both have spacing of 1σ and random initial speeds between 1 and 0.

3.2 Linked cell method

The lc_space class divides the whole space into smaller boxes of cutoff length 2.5σ. In the force calculation loop, only particles in the same and in neighboring boxes are considered. After every time step, the list which stores which particle is in which box has to be updated. This makes the whole simulation scale as $O(n)$.

3.3 Buffered force functor

To avoid recalculating the value of the force at a certain distance for every particle and to avoid calculating the square root of the distance between two particles, a buffered force functor was implemented. Upon initialization it stores the value of the force for 6000 distances between zero and cutoff in a vector. Its operator()(x) returns force($\sqrt{x}$) / $\sqrt{x}$.

3.4 OpenMP parallelization

To reach maximum performance I experimented with OpenMP to parallelize the force calculation. I used an OpenMP *parallel for* for the outermost loop of the force function. This means that the total number of boxes in the space is divided between the different threads. Using OpenMP, a speedup factor of 2 could be gained with 4 threads.

4 Results

4.1 Energy constancy

Figure 1 shows the evolution of the total energy with time. The simulation included 250 particles over 1000 timesteps. The only difference between the two lines is that for the red line all particles in space were considered to calculate force and potential. In the other case linked cells were used. All the rest like force and potential functors were the same.

If one runs the simulation for longer time, no more bumps in energy appear in the linked cell version, i.e. the error made by only considering particles in neighboring boxes seems to depend on the density of the particles.

Previously a time step value of $\Delta t = 0.0056$ was used, but for $\Delta t = 0.00188$ the precision in total energy achieved was much higher.

The mistake in the previous version lay in a wrong (constant) prefactor for the kinetic energy.

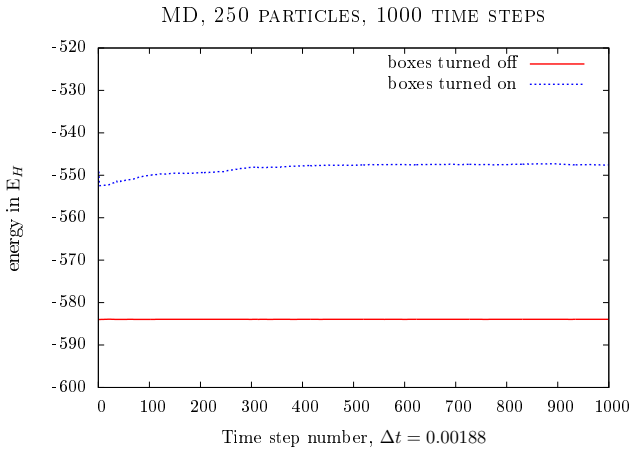

Figure 1: Evolution of system energy with time

4.2 Movie

For generating a movie, the simulation was initialized with a 2D grid of 25 particles. After every timestep a frame was output and later converted into an image with gnuplot and assembled to a video with mencoder.

4.3 Performance

The simulation is able to propagate 25000 particles over 1000 time steps in 8.3s (serial), or in 4.3s (OpenMP, 4 threads). The intermediate frames were not written to disc, but to a RAM tmpfs in memory. The $O(n)$ scaling with the number of particles could be experimentally well confirmed, thus if one would run the simulation for 1 day over 1000 time steps, one could simulate 502 million particles (with OpenMP activated).