ETHzürich **High Performance Computing for Science and Engineering I**

P. Koumoutsakos, M. Troyer
ETH Zentrum, CLT F 12
CH-8092 Zürich

Fall semester 2014

Set 9 - Molecular Dynamics and Cell Lists

Issued: November 21, 2014
Hand in: November 28, 2014, 8:00am

In this exercise sheet we will write a molecular dynamics solver for N particles interacting via a Lennard-Jones $V_{LJ}(r)$ potential (fig. 1) in two dimensions. The position $\vec{x}_i$ and velocity $\vec{v}_i$ of a particle i is govern by Newton's equation of motion

$$\frac{\mathrm{d}\vec{x}_i}{\mathrm{d}t} = \vec{v}_i \tag{1}$$

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = \vec{a}_i = \frac{1}{m_i}\vec{F}_i\,, \tag{2}$$

where $\vec{F}_i$ is the sum of all forces acting on the particle of mass m_i.

To integrate these coupled ordinary differential equations we will use the popular Velocity Verlet algorithm

$$\vec{x}_i(t+\Delta t) = \vec{x}_i(t) + \Delta t \cdot \vec{v}_i(t) + \frac{1}{2}(\Delta t)^2 \cdot \vec{a}_i(t) + \mathcal{O}((\Delta t)^3) \tag{3}$$

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t) + \Delta t \cdot \frac{\vec{a}_i(t) + \vec{a}_i(t+\Delta t)}{2} + \mathcal{O}((\Delta t)^2)\,. \tag{4}$$

We will calculate the forces in a simple N^2 loop and concentrate on optimizing the force calculation for the individual particle pairs in the first exercise. In the second exercise we will improve the scaling of the loop by introducing cell lists.

Question 1: Molecular Dynamics for the N-body problem

Here we will implement the Verlet algorithm for the N-Body problem. For simplicity you may assume all particles to have identical masses and set them to unity. As an (optional) starting point we provide a skeleton code `nbody_skeleton.cpp`.

A solution based on the provided skeleton is provided in `nbody_serial_nocells.cpp`. The solution code also provides different initial conditions (e.g. homogeneous square lattice) which you can chose from by calling the program with an additional argument. The solution code also assignes random initial velocities according to a given mean kinetic energy which needs to be provided as another additional program argument. To run the solution for the same setting as given in the skeleton code you have to execute
`./nbody_serial_nocells 1.0 100 0.05 5.0 1e-7 1000000 1000 circle 0`.

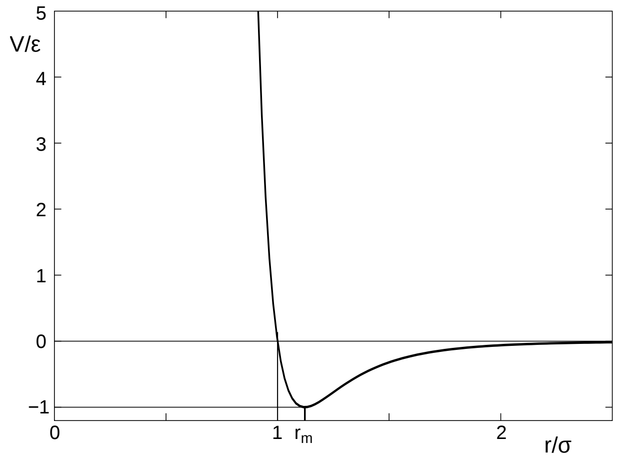


Figure 1: The Lennard-Jones potential (from `http://en.wikipedia.org/wiki/Lennard-Jones_potential`).

a) Write a function that, given the positions of two particles at positions $\vec{x}$ and $\vec{y}$, calculates the Lennard-Jones force

$$\vec{F}_{LJ}(\vec{x},\vec{y}) = -\nabla_{\vec{x}} V_{LJ}(\vec{x},\vec{y}).$$

The Lennard-Jones potential V_{LJ} only depends on the relative shortest distance of the two particles with respect to the periodic boundary conditions $r = |\vec{r}| = |\vec{x}-\vec{y}|$:

$$V_{LJ}(r) = \varepsilon\left(\left(\frac{r_m}{r}\right)^{12} - 2\left(\frac{r_m}{r}\right)^{6}\right), \tag{5}$$

where the parameters ε and r_m control the depth and the width of the potential.

Truncate the potential at a radius r_c smaller than the diameter of the simulation box and impose periodic boundary conditions. Your code will spend most of its time in this function, so try to make it fast. Explain what optimizations you implemented.

Adding up all forces acting on particle $i = 0$ using the initial conditions of the skeleton code and the parameters given in equations 6 you should obtain

$$\vec{F}_0 = \vec{a}_0 \approx (0, 120849)^{\mathrm{T}}.$$

The formula to be implemented is

$$\vec{F}_{LJ}(\vec{x},\vec{y}) = \begin{cases} 12\epsilon\left(\frac{r_m^{12}}{r^{13}} - \frac{r_m^6}{r^7}\right)\frac{\vec{r}}{r} & \text{if } r \le r_c\,, \\ 0 & \text{if } r > r_c\,, \end{cases}$$

and r_c is the cutoff radius.

The most important point here is that no expensive `sqrt` or `pow` operations are necessary. After that, one can minimize the number of operations by taking care to calculate common factors only once. Apart from a single inversion of the squared radius one can get away with only additions and multiplications. If the function uses loops over coordinates, the number of iterations should be known at compile-time so the compiler can unroll them.

b) Use the force calculation to implement a time step of the (velocity) Verlet algorithm for a system of N point particles.

If our skeleton code is used one needs to implement the `update_positions` and `update_velocities` functions according to the formulas given in the lecture slides. The only part not given there explicitly is enforcing periodic boundary conditions.

c) Write a function that measures the potential and kinetic energy of a configuration.

$$E_{\text{pot}} = \sum_{i \neq j} \left(V_{LJ}(\vec{x}_i, \vec{x}_j) - V_{\text{shift}} \right), \qquad E_{\text{kin}} = \frac{1}{2} \sum_i m_i \vec{v}_i^2 .$$

Note: Due the cutoff at radius r_c we introduced a discontinuty in the potential $V_{LJ}(\vec{x}_i, \vec{x}_j)$. We compensated for this by shifting the potential by $V_{\text{shift}} = V(r_c)$.

The formulae are implemented in `simulation::measure_energies()` and `potential::operator()(skalar_type r2)`.

d) Put the pieces together and test your code: Choose a set of particle positions and velocities $\{\vec{x}_i, \vec{v}_i\}$ as initial condition and evolve the system in time. Plot the particle positions for several time steps and check that the time evolution looks physically plausible. Monitor the kinetic and potential energies and check that the total energy $E = E_{\text{pot}} + E_{\text{kin}}$ is conserved during the simulation. A good set of parameters are given in the invocation example of the skeleton code:

$$\text{box} = [0, 1[^2, \quad n_{\text{particles}} = 100, \quad r_m = 0.05, \quad \varepsilon = 5.0, \quad \Delta t = 10^{-7} \tag{6}$$

with a cutoff radius $r_c = 2.5\sigma = 2.5 \cdot 2^{-1/6} \cdot r_m$.

You may also use the `animated_configurations.py` plot script to plot the time evolution of the particles. Running the simulation for 100000 time steps and printing all 1000 time step should give you a nice little movie.

If the force calculation produces infinities you may need to reduce the time step by an one or more orders of magnitude.

With the given parameters of the Lennard-Jones potential, 100 particles in the unit square give a reasonable density with a reasonable high potential energy. The circle configuration produced by the skeleton's `init_circle` has $E_{pot}/N = 4375.58$.

For these parameters, the system starts with high potential but no kinetic energy. After a short period of time the system equilibrates and converts a large part of the initial potential energy into kinetic energy resulting in a fast movement of the particles. After an initial equilibration the kinetic and potential energies fluctuate around their mean values. The total energy is conserved up to 4 digits.

e) Measure the scaling of the serial code with the particle number.

The scaling is shown in Fig. 3. As expected the simple nbody solver is dominated by the $O(N^2)$ scaling of the force calculation. Note that we did not use the circle as initial condition here, because the potential energy for this configuration will increase dramatically when adding more particles (due to the small distance between neighbors) and the simulation will become unstable. Instead we aranged the particles on a square lattice filling the full domain and gave them an initial kinetic energy of $\bar{E}_{kin}/N = 1000$.

Question 2: Cell Lists

a) Extend your serial implementation of the MD code from the previous question to use cell lists. Comment on the theoretical advantages and measure the scaling of the algorithm with particle number. Compare to your previous results.

   For implementing the cell lists, we replace the single vector of particle positions `x` by a vector of vectors, namely one vector per cell, and likewise for velocities and accelerations. After updating the positions we need to run over all particles and put those that have moved over a cell boundary into their new cells. Moving a single particle requires constant effort so the complete sorting is $O(N)$. We choose to label the cells sequentially within each row allowing for a simple and fast index computation and ensuring data locality at least along one dimension.

   When the particles are sorted into cells, the inner loop of the force calculation can be restricted to particles in the same cell as the first particle and its neighboring cells, hence reducing the cost of the dominant force calculation substantially.

   At first glance the scaling with increasing particle number does not improve over the simple force calculation (fig. 3, left). This can be understood if one considers the number of cells used. By just increasing the number of particles while keeping the system size constant the number of particles in each cell will increase, but the number of cells will remain the same. Within each cell we still have an $O(N^2)$ loop to compute the forces. Hence, the same scaling is actually expected in that case.

   However, if we increase the system size and the particle number in the same manner, i.e. keeping the density constant, the mean number of particles per cell remains the same, resulting in constant cost for the inner force calculation loop. Therefore the overall complexity of the simulation is reduced from $O(N^2)$ to $O(N)$. The right side of fig. 3 clearly shows this difference in scaling. The speedup rises from a factor of two with only $N = 100$ particles and 4^2 cells to fifty for $N = 5000$ and 31^2 cells, proving once more that the choice of algorithm is often more crucial for performance than optimization of the code.

b) Parallelize your code using OpenMP and explain your parallelization strategy.

   We parallelize the outermost force calculation loop with OpenMP, which is the loop over cells. For small systems this will lead to load balancing issues as there is only a small number of cells. We cannot trivially improve this with the `collapse` option of `omp for` as the loop over particles within a cell has different length for different cells. For large systems there should be no problem however.

c) Show the strong and weak scaling behavior of your code up to 24 cores.

   The scaling of the total runtime with the number of threads (Fig. 4) saturates rather fast. This is understood when remembering that the complete effort is $O(N)$ and may parts of the code become relevant. Since a significant part (e.g. the cell sorting) remains serial, Amdahl's law will yield a low maximal speed-up. Parallelizing these parts may overcome the problem at least partially.

Summary

Summarize your answers, results and plots into a PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.

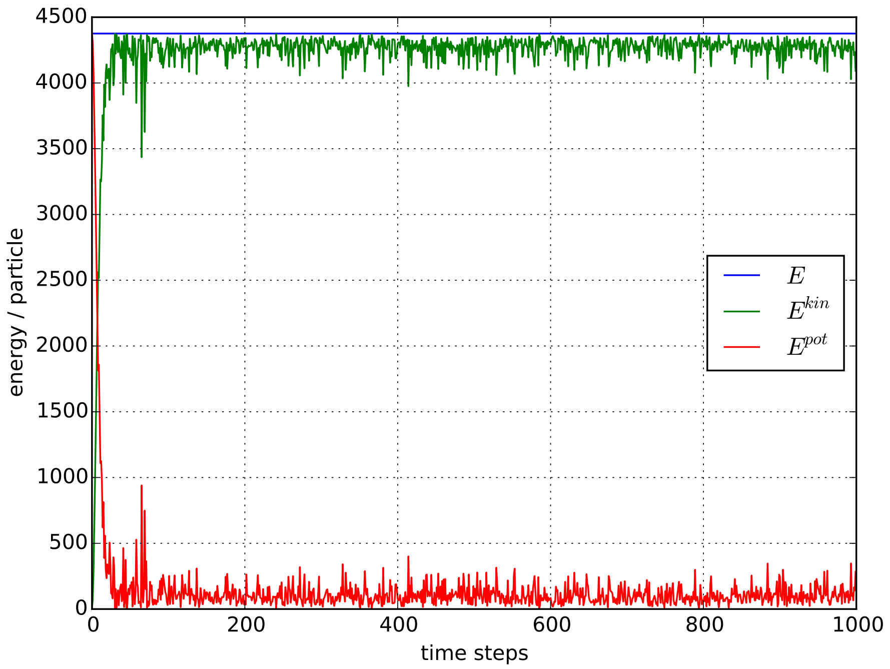


Figure 2: Total, kinetic and potential energy per particle for the first 100 000 time steps measured each 1000 steps in the evolution of 100 particles.

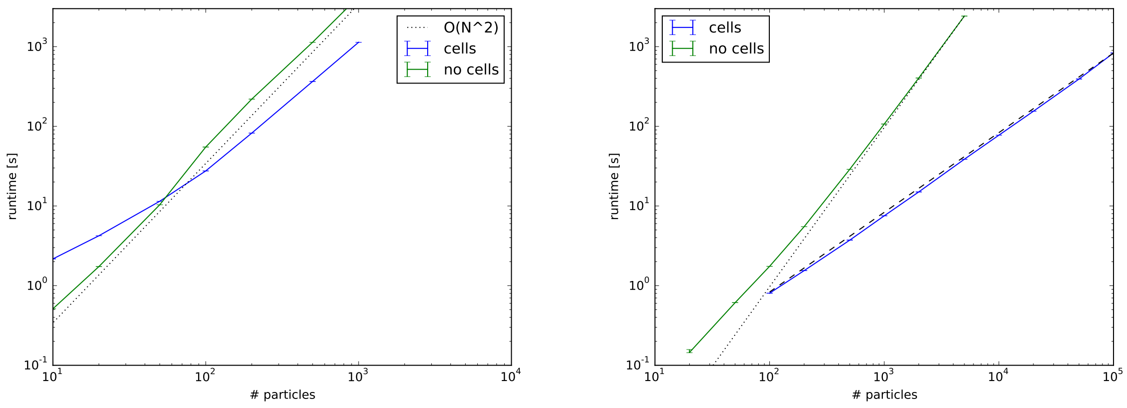
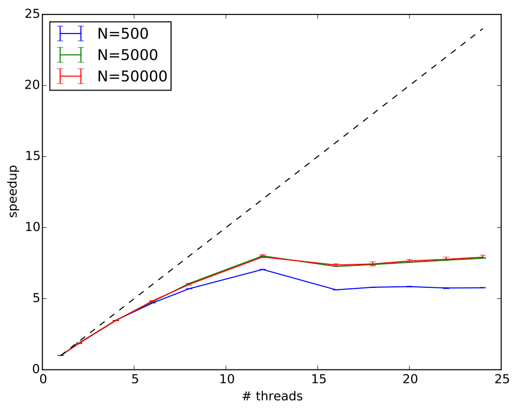


Figure 3: Serial execution time for different number of particles, while keeping the system size constant (left) and while keeping the density constant (right). While for a constant system size both methods scale $\sim N^2$ (indicated by dotted line) with the number of particles N, for constant density N/V the implementation with cell lists scales linearly in the number of particles ($\sim N$, dashed line) whereas the naive implementation scales $\sim N^2$ (dotted line).



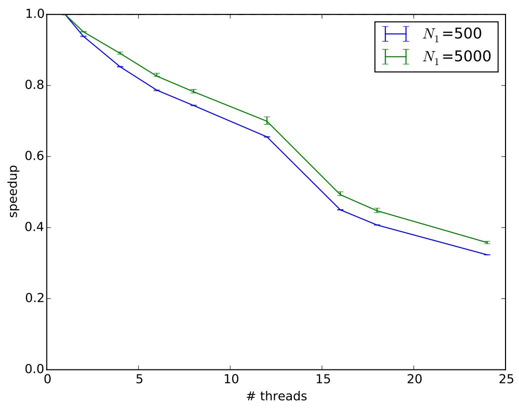


Figure 4: Strong (left) and weak (right) scaling plots for MD simulations with N particles. For the weak scaling analysis, the number of particles is increased linearly with the number of threads.