Project 1: N-body simulation

Motivation

N-body simulations are common in physics and chemistry. To learn the basic model we use the $O(n^2)$ complexity model here where each body is updated by the applying the forces of all other bodies. The approach is infeasible for very large values of n, and is thus usually only used for systems where n is small and the pairwise force-calculation is essential. The actual simulation will focus on the Nice model[1], where the primary interest in in the behavior of asteroids in a solar system. Here the mass of asteroids is so small that we need not consider their gravitational pull on the sun, planets or even other asteroids. This means that we can model a sun, a small number of planets and a large number of comets even on a single computer. The complexity of the Nice algorithm is $O(n^2+n*m)$, where n is sun+planets and m is the number of asteroids. The system shown below has one sun, nine planets and 1000 asteroids. The actual simulation you will be performing will include 1e4, 1e5, and 1e6 asteroids.

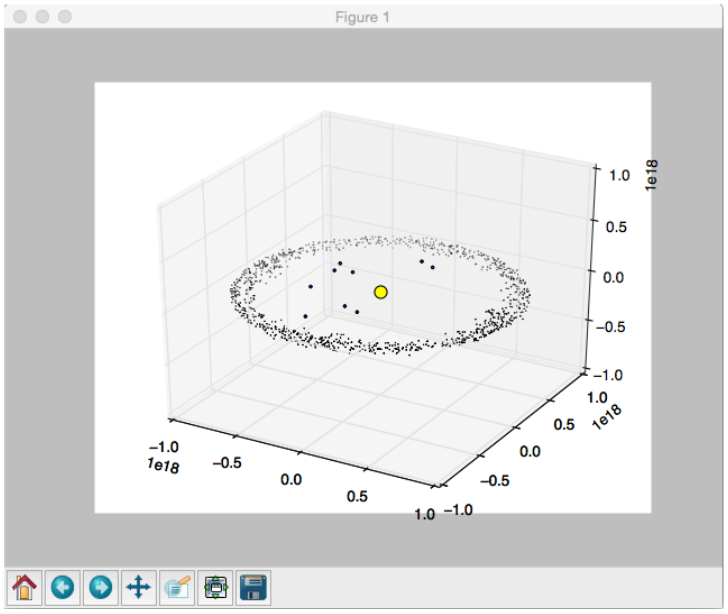

Figure 1 Nbody Nice simulation

The bodies are represents individually as:

```
{'m': 10.0, 'x':1, 'y':2, 'z':3, 'vx':0, 'vy':0, 'vz':0}
```

which means a body with mass 10.0 placed at (1,2,3) and velocity (0,0,0).
The code uses only Newtonian forces and a leap-frog progression, i.e.:

$$F^t = \frac{m(v^{t+\frac{1}{2}} - v^{t-\frac{1}{2}})}{\Delta t}$$

$$x^{t+1} - x^t = v^{t+\frac{1}{2}}\Delta t$$

where F is the force on a body, v is the velocity and x is the position.

Thus be sequential code with our original representation of the bodies becomes:

```
def calc_force(a, b, dt):
    """Calculate forces between bodies
```

[1] http://en.wikipedia.org/wiki/Nice_model

```
    F = ((G m_a m_b)/r^2)/((x_b-x_a)/r)
    """

    r = ((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2 + (a['z']
         - b['z']) ** 2) ** 0.5
    a['vx'] += G * a['m'] * b['m'] / r ** 2 * ((b['x'] - a['x']) / r) \
        / a['m'] * dt
    a['vy'] += G * a['m'] * b['m'] / r ** 2 * ((b['y'] - a['y']) / r) \
        / a['m'] * dt
    a['vz'] += G * a['m'] * b['m'] / r ** 2 * ((b['z'] - a['z']) / r) \
        / a['m'] * dt


def move(a, b, dt):
    """Move the bodies
    first find forces and change velocity and then move positions.
    For sun and planets calculate all pairs, for asteroids calculate each
    Asteroid against every sun and planet.
    """

    for i in b:
        for j in b:
            if i != j:
                calc_force(i, j, dt)
    for i in a:
        for j in b:
            if i != j:
                calc_force(i, j, dt)

    for i in a+b:
        i['x'] += i['vx'] * dt
        i['y'] += i['vy'] * dt
        i['z'] += i['vz'] * dt
```

Considerations on the parallel version

This task is a classic example of a problem where the solution becomes simpler by using more memory. For all pairs we need to calculate the distance, multiply by weights and apply the forces. It is easily seen that this can be done using vectors, i.e.:

```
for i in b:
    for j in b:
        if i != j:
            calc_force(i, j, dt)
```

It is simple to see that if galaxy is represents as vectors rather than individual bodies you can eliminate the inner loop. You can however, also represent the bodies as both the row and column in a matrix and the body of the matrix be the relation between the two bodies. In that way you eliminate the loop entirely. The diagonal will represent the body onto it self – if that is included you will get a distance of zero and thus an error, so this must be eliminated. The same problem obviously does not exist for the asteroids.

Programing Task I

The solution should be implemented using vectorization, and based on basic Newtonian physics. Your implementation should transform the provided loop-based version into a vectorized version in NumPy. Note that the first, and largest, part of the assignment is to change the datastructure from the records of data for each body and into vectors where all x, y, z,… values are kept in one variable.

Report

Your report, which should be submitted through Absalon, should be no longer than 3 pages in total. You should explain how you have transformed the loops into vectorized code and how this has influenced your performance. You should provide experiment results with 1e4, 1e5 and 1e6 bodies, each with 10 time steps, where you compare the performance of the original version to your vectorized solution. The results should be presented as an easy the read graph, which includes the absolute and relative before and after transformation of the code, as an example below.

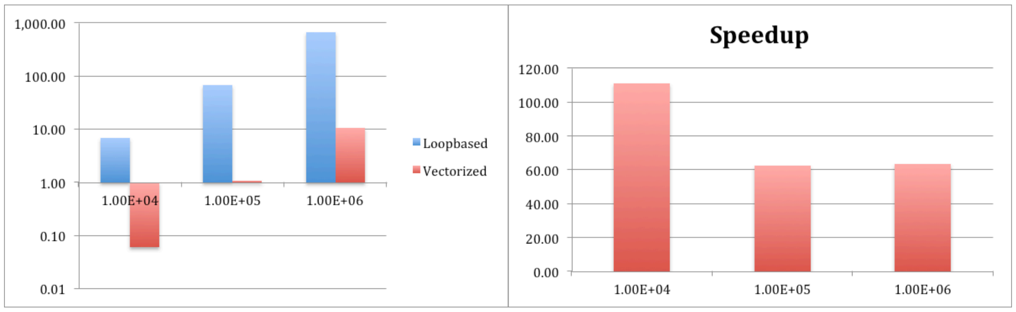

Figure 1 Examples of speedups from vectorization of the Nice code – not from the hardware you will be using!

To save resources you should not run the original version on the cluster but instead use the below numbers for your sequential baseline. You should run your vectorized code on “Marge”[*] as the mRSL file provided for you also specifies.

Bodies	Run 1 (sec)	Run 2	Run 3	Average
1e4 bodies	16,21	16,49	16,25	16,32
1e5 bodies	161,89	165,88	163,16	163,65
1e6 bodies	1616,78	1658,22	1627,89	1634,30

Table 1 Loop-based data for the Nice-code, run on “Marge”

[*]Marge is a 2 x 4-core Intel Xeon E5640@2.67GHz, with 96GB memory.