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.

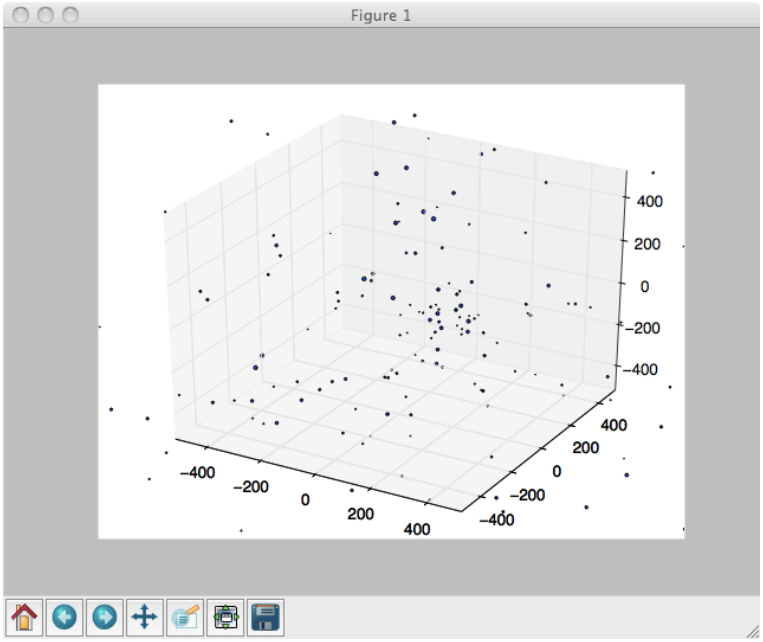


Figure 1 N Body simulation

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
    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(galaxy, dt):
    """Move the bodies
    first find forces and change velocity and then move positions
    """

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

    for i in galaxy:
        i['x'] += i['vx']
        i['y'] += i['vy']
        i['z'] += i['vz']
```

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 galaxy:
    for j in galaxy:
        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.

Programing Task I

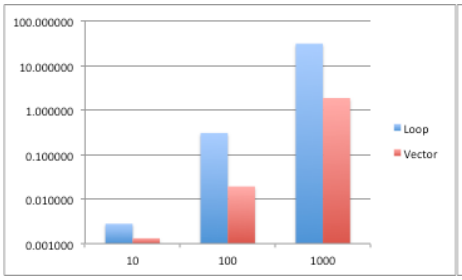
The solution should be implemented using vectorization, and based on basic Newtonian physics. Your implementation should transform the provided loop-based version into a vectorized version in NumPy.

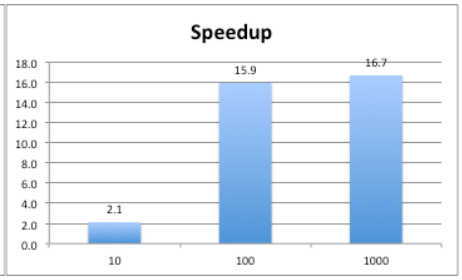
Programing Task II

Once you have successfully completed the vectorization version in NumPy port it to use a GPGPU using PyOpenCL and run the simulation on a GPGPU.

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 10, 100 and 1000 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.


100.000000
10.000000
1.000000
0.100000
0.010000
0.001000
10
100
1000
Loop
Vector


Speedup
18.0
16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
2.1
15.9
16.7
10
100
1000