**N-Body Problem**

**Documentation**

1. **Description**

This problem is derived directly from Newton’s Principles.

Statement: given the current position and velocity of N Astronomical Bodies and using Newton’s Principles we should be able to predict the positions of those bodies at any given time, past or future

Unfortunately, exact analytical solutions exist only for the case when 2 bodies orbit each other

There are also exact analytical solutions for certain corner cases, such as periodic orbits, negligible mass for the third body etc.

This is caused by the fact, that such systems are governed by chaotic dynamics, a small change in the initial conditions being able to create multiple possible outcomes

Thus, approximate solutions are used in the industry to correctly plot trajectories for rockets, satellites and other such things. These approximate solutions rely on numerical integration to arrive at almost correct results.

1. **Approach**

To take into consideration when trying to solve this problem:

* the forces (gravity) exerted by each body on all other bodies (i.e. same values, reverse direction of all force vectors applied to the current body)
* collisions
* inertia (property of mass)
* tidal waves, they have a negative impact on velocity
* radioactive decomposition; this has to be taken into account only when t (time) tends towards infinity; otherwise negligible
* relativistic mechanics; due to the fact that gravity curves space-time, basically all orbits are conic sections of a sphere

1. **From a programming point of view**

All data is read from an input file. There will be a certain number of bodies, and a number of time steps. For each of those time steps, called cycles hence forth, the properties of the bodies such as position, velocity and acceleration (all 3D vectors) will be updated.

This update takes place with regards to inertia, collisions and the gravity applied by the other bodies to the current body.

* 1. **Multithreaded**

The number of threads can be configured at startup from a properties file.

The program creates the threads; each of these will read their own slice of the input file in order to get their corresponding bodies, and then put them in the global bodies list. The access for writing to the global bodies list is synchronized in a synchronized block (mutually exclusive). After these reads are done, all threads will wait at a count down latch, in order for all other threads to finish reading their input.

Then, processing can begin. For each time step, each thread will update the data of its own bodies with regards to the other bodies. In order for concurrent access over the bodies objects to be successful, each body has associated with it a ReentrantReadWriteLock, which will be used for readLocks in getter methods and writeLocks in setter methods.

At the end, the main thread will wait for all other threads to finish their work, and will attempt to close the executor service.

1. **Stats**

Data: 6000 bodies, 100 time steps, intervals of 1000 bodies for each thread

Observations: the optimum number of threads on my resources is at around 24; the higher the number of bodies in the system, the more performance we gain with the multithreaded approach. Idea: would be nice to try it on a nebula (~ millions of bodies)

|  |  |  |
| --- | --- | --- |
| Approach | Number | Time (ms) |
| Serial | - | 1556067 ~ 26 minutes |
| Threads | 8 | 451581 ~ 7.5 minutes |
| Threads | 24 | 368407 ~ 6 minutes |
| Threads | 36 | 371783 ~ 6.1 minutes |

1. **Resources**

<https://www.youtube.com/watch?v=et7XvBenEo8>

<https://www.britannica.com/science/celestial-mechanics-physics/The-n-body-problem>

<https://rosettacode.org/wiki/N-body_problem#C.23>