N-Body Problem

- Explanation
- Approach
- Parallelisation
- Algorithm-Simulation
- Conclusion

10-10-20
CED17I047
Ritvik Raj Singh

Problem Formulation:

The N-body problem is one of famous problems in classical physics for predicting the motion of n celestial body that interacts gravitationally in free space.It is a problem for predicting individual motion of bodies starting from a quasi state.

The problem has been motivation to understand motions of sun ,planets and other celestial bodies in global clusters . Consider general relativity the problem is difficult to solve and still is an open problem .See two-body problem and restricted three-body problem which have been solved .

The below problem solution is based on problem of simulation of random 10000 masses ranging from 34000,25000000 kg on **2d** plane on (-1000,1000) on both x and y coordinates where initial states of bodies are in quasi state (initial velocity and acceleration are 0 in both x axis and y axis) and initial position are ((-500,500)|(-400,600)) in x and y axis respectively .

Assumption:

For simulation purpose and ease of calculation classical newton laws are used for computing velocity and acceleration of individual bodies .

fi,j(t) =−Gmimj||ri(t)−rj(t)||3(ri(t)−rj(t))

Where i and j the body are applying f(i,j) force on each other. (Newton 3rd law) where G =6.67 × 10^{-11} Newtons kg^{-2} m^2
NOTE: While calculating position and velocity of actual planetary motion Newton Laws are no longer valid ,due to fixation of barycenter.(https://en.wikipedia.org/wiki/N-body_problem)

Approach

Assuming Classical Newton laws to hold true following formulas are used for calculating for velocity and acceleration .

$$v = u+at \quad s = ut+(0.5)a(t)*t$$

u : initial velocity
v : final velocity
a : acceleration
t : Time
s : Distance

Assuming Classical Newton laws to hold true following formulas are used for calculating for velocity and acceleration.
The simulation is based on updating the vector and velocity information at delta t timestep instead of continuous time simulation(deltaT≈0) .To make the simulation smooth lower the deltaT value.

The input for the given problem is given by coordinates of mass(ranging in 3400-2500000kg) given in file text.
The algorithm steps are:
For each time step
    Compute force computation on ith body by all n-1 body :
        Compute Vector position of j th body wrt to ith body in both x and y axis
        Computer rvector = (sqrt(dx*dx+dy*dy))^(1.5)

        $F = (Gm_i m_j * vec) /(rvector)$
        Compute acceleration on each i,j body and thus compute i,j velocity value of both i and j thus new r vector position of i with new velocity.

    Once the value of force is computed for each body then update the new position of each body simultaneously .

Repeat until time finished

Parametres:

Time step incremented by deltaT
NOTE:Varying deltaT and making it small(1-5) will make simulation smooth but computation time taken will be extremely large for number of bodies

Number of Bodies: Increasing number of bodies (maximum 111002) will increase computation but make simulation uniform .

Total Time : The number of timestep until which the simulation will run .
NOTE:Increasing the total time will increase the runtime of simulation.

OpenGL is used for simulating the bodies (look into simulation.cpp)
NOTE: The simulation is attempted only when all the computation is done for all specific bodies.It is not done simultaneously when calculating individual velocity and position of bodies at each timestamp

Parallelisation with OpenMP:

The algorithm uses block distribution as symmetry of Force , F(i,j) = F(j,i) and cyclic distribution for force calculations ,since all bodies force value computation is calculated and then updated at end simultaneously .

Since OpenMP uses shared memory for parallelisation thus cyclic distribution computation parallelisation is easy.

Thus #pragma omp parallel for directive is added for force computation from all n-1 bodies and workload is distributed on static scheduling where chunks of data are equally distributed to the number of threads that are available .

To avoid race conditions each data is saved in individual rvector position ,velocity data in vector data type in c++ so that each thread in parallel constructs ensures access of one element at index .

Parallelization is done on force of n-1 body computation is independent of each other thus no memory race condition is introduced .
Fi = ΣFj (j=1:n j!=i)

```
For each particle i do:
    foreach particlej > i do
        ompsetlock(& locks[i])
        Fi(t)+=fi,j(t).
        ompunsetlock(& locks[i])
        ompsetlock(& locks[j])
        Fj(t)−=fi,j(t).
        ompunsetlock(& locks[j])
    end for
end for
```

Speed up improvement with OpenMP:

- For n = 2000 bodies simulation over 10000s total time sampled at 5 second on 4 core system taken are the following .

Threads	1	2	4
Time(s) of execution	249.406	155.456	114.925
Parallelisation Factor		0.753390054	0.7189402019

F = p/(1-p)*((T(p)-T(1))/T(1))

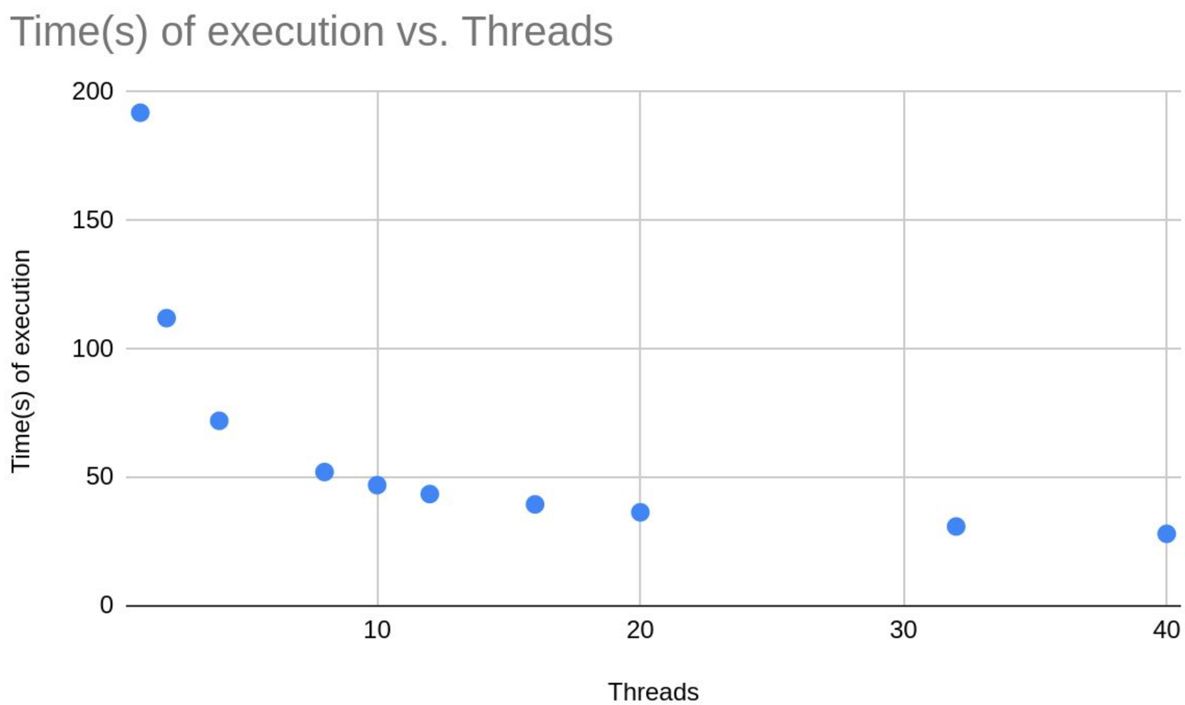
Average parallelization factor=0.73616512

- For n = 2000 bodies simulation over 2000s total time sampled at 5 second on 20 core system(max 3.00 GHZ time) taken are the following.

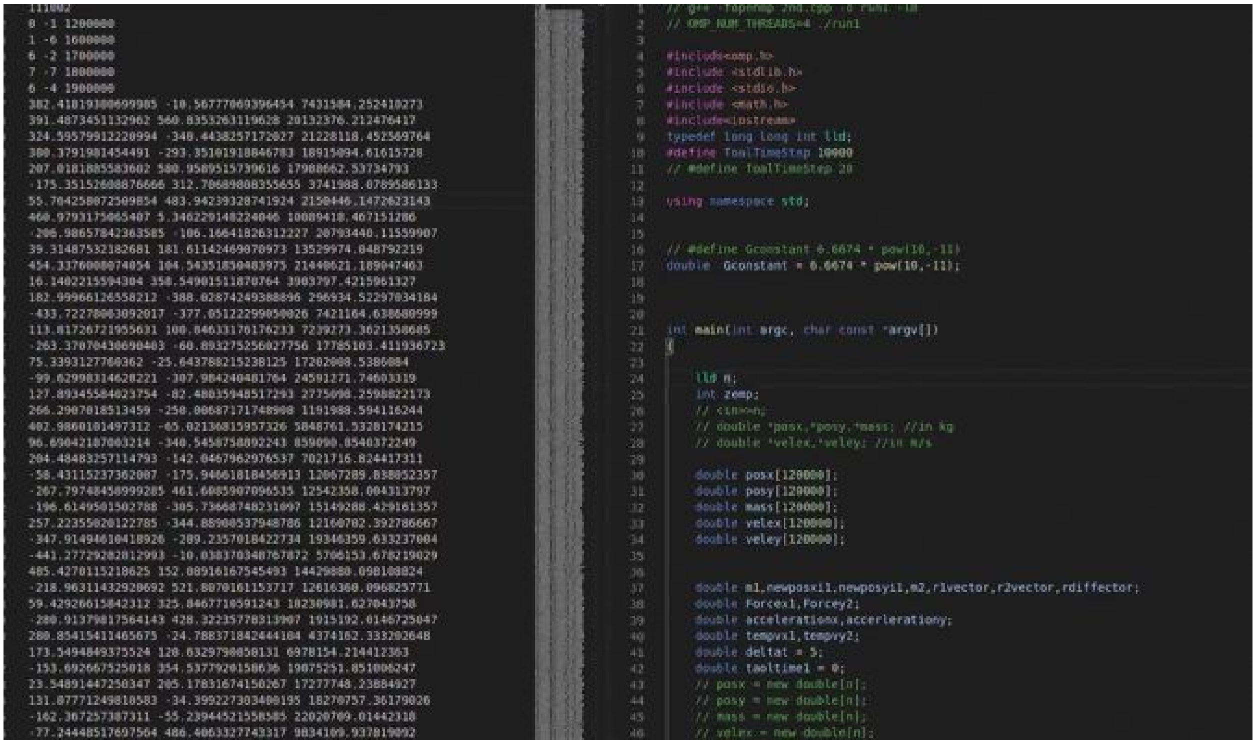
Threads	Time(s) of execution	Parallelisation Factor
1	191.981	
2	112.019	.82650996
4	72.0463	.83296622
8	52.1042	.83268240
10	46.9954	.83912007
12	43.5288	.84356188
16	39.5014	.84719275
20	36.4251	.85291280
32	30.9002	.86611115
40	28.041	.87583453

F = p/(1-p)*((T(p)-T(1))/T(1))

Average parallelization factor=0.8463213


Time(s) of execution vs. Threads
200
150
100
50
0
10
20
30
40
Time(s) of execution
Threads

Algorithm Simulation:



Check the video at
https://drive.google.com/file/d/1LtMBHbnq3cejyO7lR0vORSjGo6JRp3tB/view?usp=sharing

Run the source code .For simulation run opengl freegludev is required in ubuntu .

Conclusion:

The distributed workload decreases the time taken by almost 45 % workload .

On average the parallel section is 75 % of code .

Even though shared memory is used no memory race condition is encountered .

Equally proc load is distributed by n threads specified during runtime .