Highly Parallelized N-Body Simulation

Hyeyun Jung & Lydia Ye

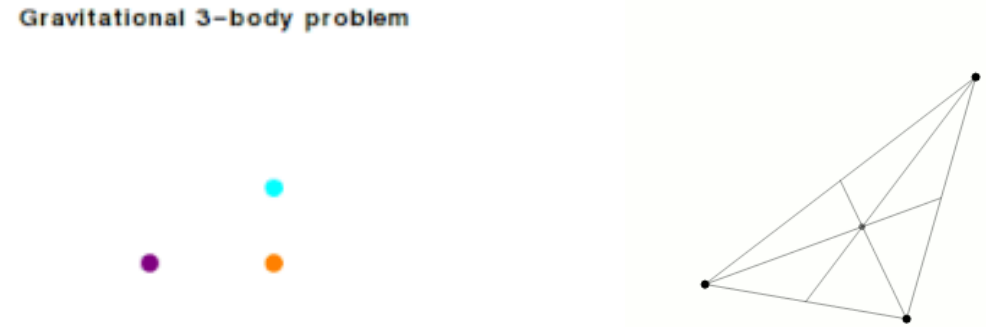


Image From: https://en.wikipedia.org/wiki/Three-body_problem

Project Introduction

Motivation

What is N-Body problem?

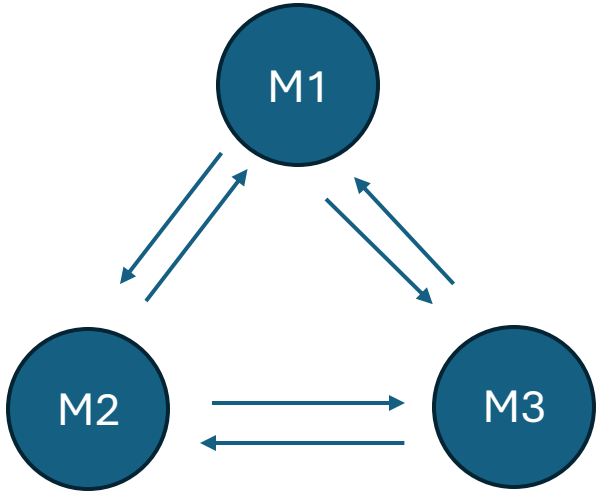
- Model gravitational interactions between objects in space.
- For N>2: Every change in position affects all other objects' forces and movements → impossible to find a closed form mathematical formula that predicts their exact positions over time.
- 2-Body problem can be easily solved using Newton's law & calculus.

Problem : Computational Complexity

- The number of calculations grows quadratically as N increases:
- Example 1: Solar System (N=9): (9 choose 2) 36 gravitational interactions.
- Example 2: Small Galaxy Cluster (N=1000): ~500,000 interactions.
- Approx (O(N^2))

Project Introduction

Motivation

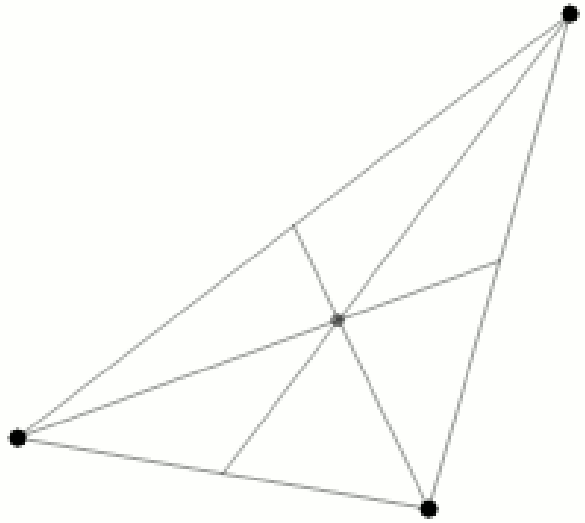


Change in one body affects others

s betwee
ition affe
ible to fin
over time.
solved u

mplexity

ows quad
9): (9 cho
ster (N=1



nat

https://en.wikipedia.org/wiki/N-body_problem

Project Introduction

Goal

Make Faster

- Create a **highly parallelized** simulation of the N-body problem
- Leverage GPU parallelization for large-scale simulations (N>1000)

Visualize

- Draw each object’s position at each time

3 Concepts

Concepts used

Parallelism with GPU

- Use CUDA to accelerate pairwise force calculations across N bodies

Thread Synchronization

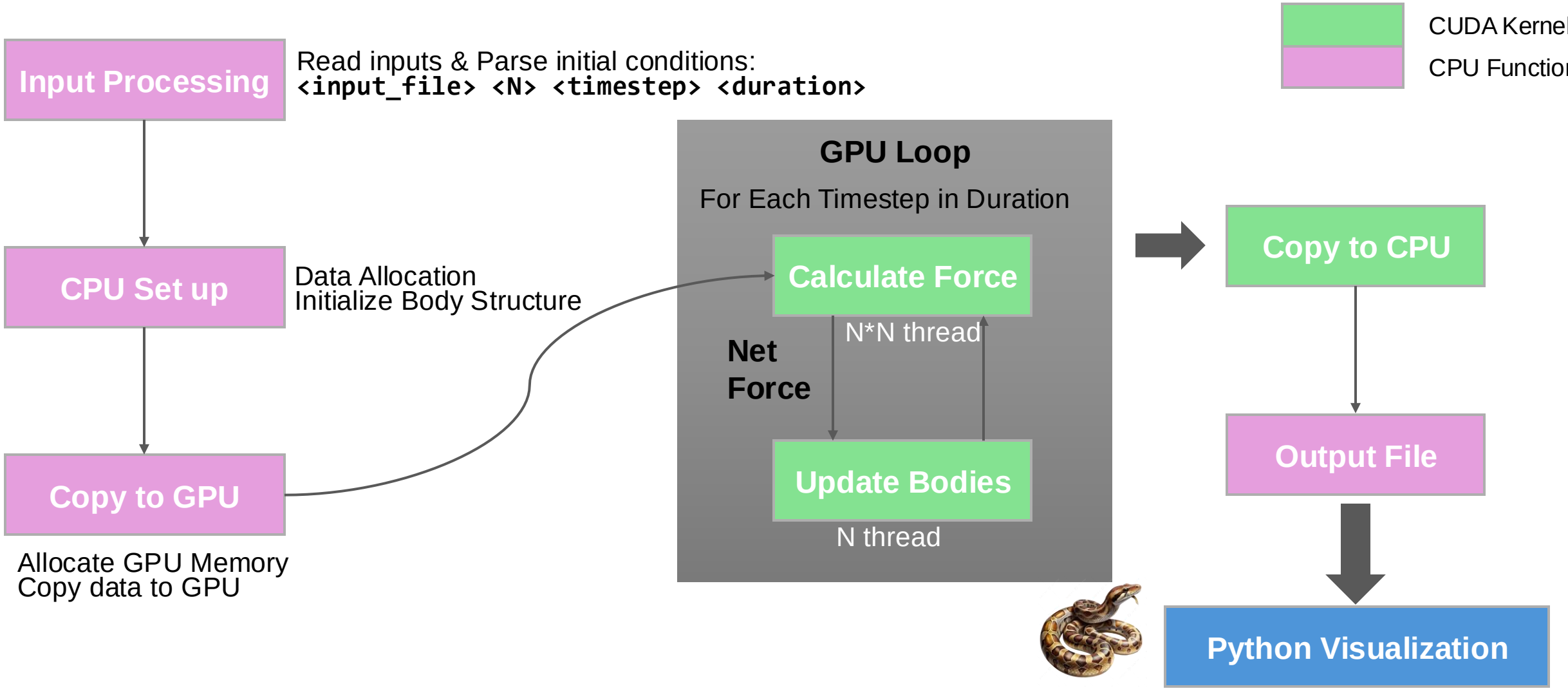
- Share intermediate results on GPU’s shared memory

File System

- Get input data from and output results to CSV files

Implementation

Overview



Implementation

Challenge 1

- The celestial objects have extreme values like 10^24, 10^30 and gravitational constant is power of 10^-11. The extreme values often result in incorrect calculation.
  - Decided to use normal(?) values (G= 0.1)

Challenge 2

- Managing multiple data structures between GPU and CPU (copying..etc..)
- Cannot visualize in real time since it required copying to CPU every time

Demo

csc-213-n-body > random_data.csv > data

```
97596.507812,56532.562500,-97093.703125,-63939.898438,42857.207031,-9641.415039,-99191.414062
69977.585938,14079.487305,-81894.359375,14249.277344,13376.176758,75522.375000,23200.595703
37625.574219,608.921021,-57428.929688,-74675.257812,89767.312500,65415.429688,18988.763672
1513.434692,9309.148438,-27659.005859,-62412.457031,-49494.695312,24025.677734,31530.058594
14301.583984,-2443.492432,81363.382812,-62583.187500,89624.914062,72317.625000,-88565.609375
64886.929688,-12774.634766,60694.027344,73283.156250,5056.989258,20955.158203,-44834.949219
84310.367188,44879.089844,-62232.785156,57077.457031,33695.066406,58233.703125,80081.437500
51071.421875,56966.257812,82401.539062,16845.798828,-78805.398438,60446.773438,-8277.398438
44278.214844,-55898.863281,-7841.247559,-30170.345703,-16395.132812,-49351.261719,28426.515625
27225.642578,8594.059570,80784.281250,-35767.992188,-57264.636719,43580.746094,98894.835938
17616.511719,-99888.156250,55176.640625,-54328.386719,94486.484375,-56132.019531,-79846.328125
88744.843750,-15031.647461,-71943.687500,-66366.992188,-94516.289062,64050.566406,23094.212891
77719.398438,-10698.896484,49657.023438,-67228.539062,25354.480469,68259.773438,45587.097656
94724.750000,-69247.429688,94688.304688,-58265.335938,61912.238281,-87942.382812,62879.859375
56422.171875,24286.699219,-74447.000000,69259.070312,84298.718750,99134.781250,-63925.996094
50404.523438,5041.551758,80399.500000,81896.007812,89488.437500,-39343.441406,87285.671875
22910.191406,-80446.257812,-33435.523438,5402.219238,-79057.515625,23358.142578,-9510.350586
90837.906250,70080.945312,75025.132812,31208.765625,23188.865234,-65248.804688,-24909.019531
23242.332031,74287.007812,45910.167969,-34349.066406,-44717.621094,-84033.062500,-17257.994141
48118.585938,92291.710938,47114.394531,-82847.226562,-7917.588867,23678.111328,-43756.968750
```

Generate Random Data

```
./generate_bodies 100
```

- **Purpose**: Creates initial conditions dataset for N-body simulation and visualization
- **Input**: Integer N - Defines number of bodies to generate
- **Output**: File `random_data.csv` containing:
  - Mass values for N bodies
  - Position coordinates for N bodies (x,y,z)
  - Velocity vectors for N bodies (x,y,z)

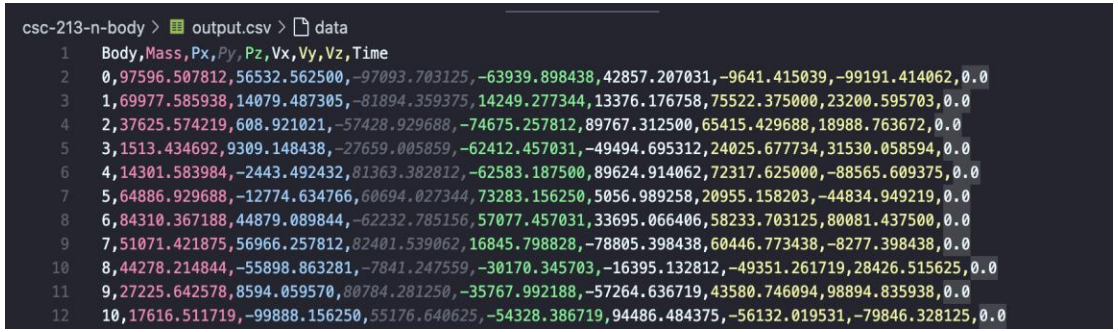
Demo

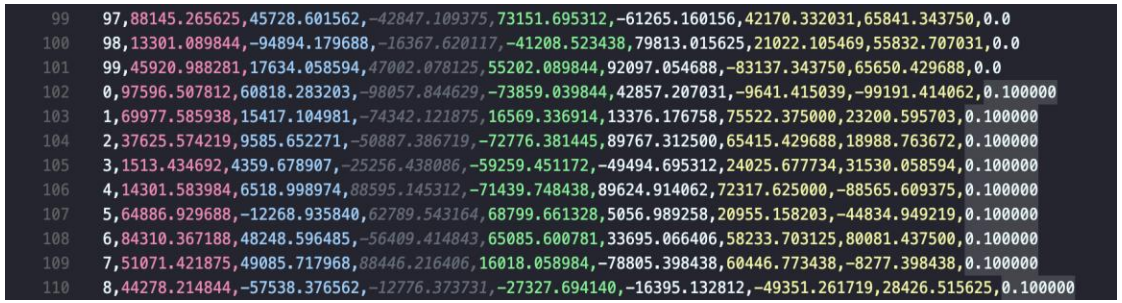
Simulation

Simulate N-Body Problem

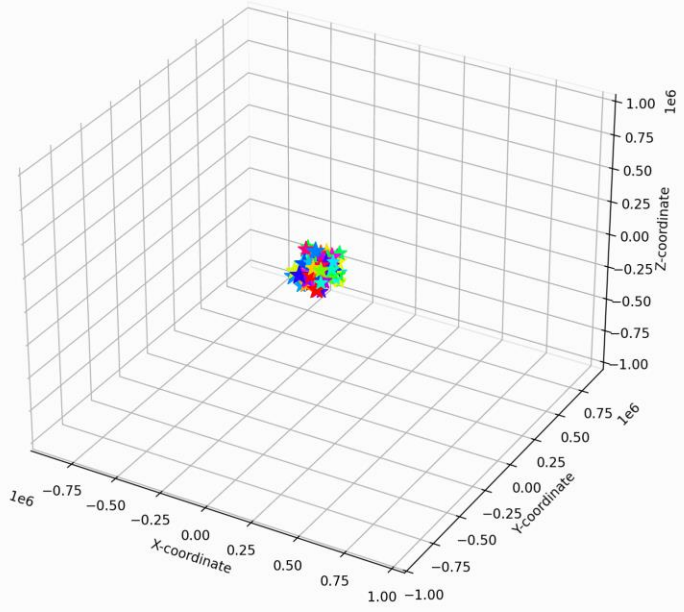
```
/simulate_n_body random_data.csv 100 1 100
```

- Input Parameters:
  - CSV file path with initial body data
  - Number of bodies (e.g. 100)
  - Timestep length (e.g., 0.01s)
  - Simulation duration (e.g. 10 seconds)
- Output: Creates `output.csv` containing:
  - Body ID, Mass, Positions (x,y,z), Velocity (x,y,z), Time step
  - Tracked at each timestep throughout simulation duration





Demo



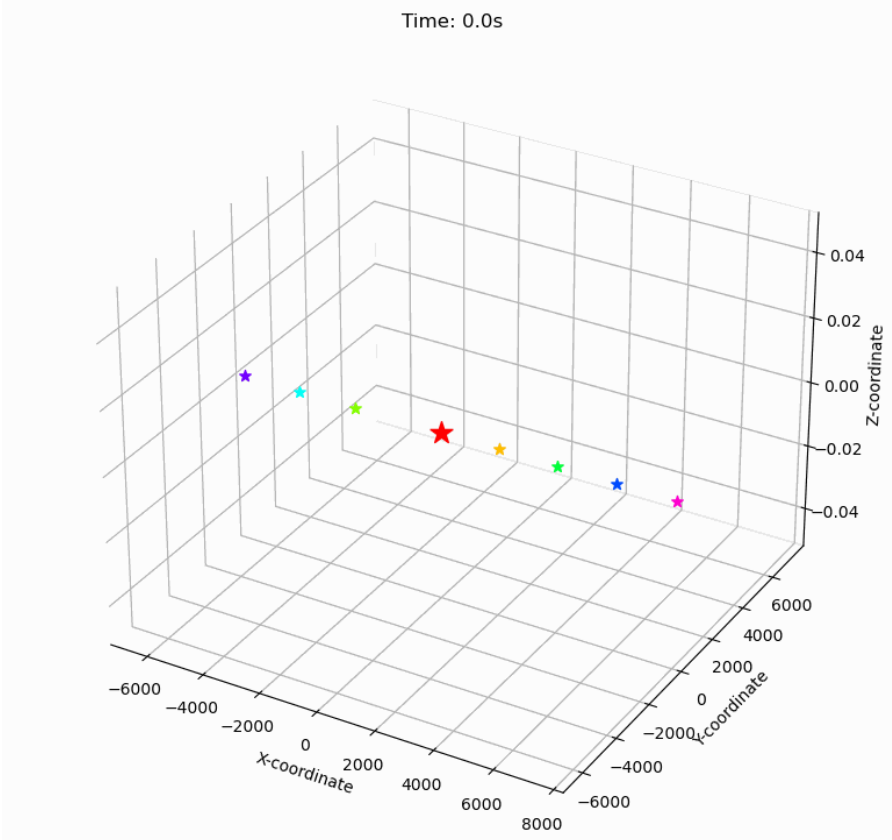
Visualizes the output data

```
python visualization.py
```

- Reads data from a local `output.csv` file
- Displays the animation to show how the position of the bodies changes throught the time steps
- Creates visualization.gif to store the animation

Demo

Mimicking Solar System

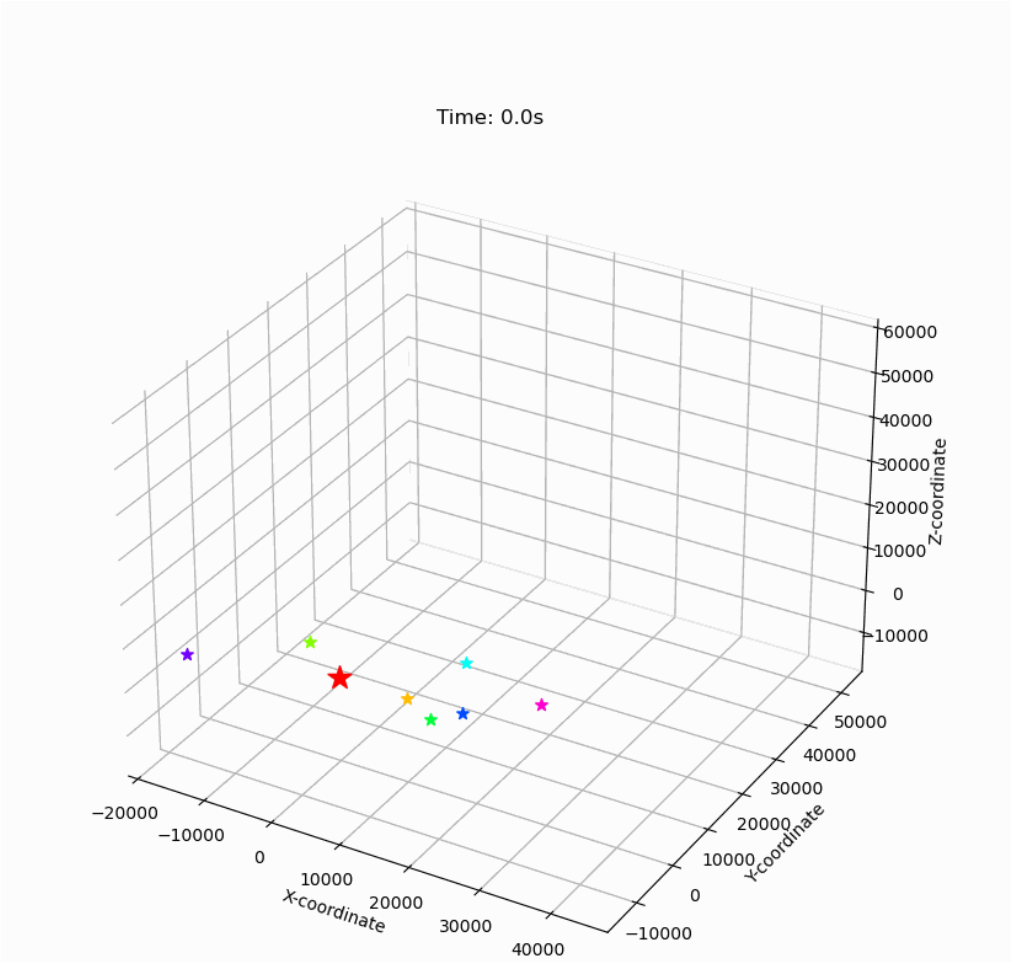



```
/simulate_n_body inputs/sample_data_1.csv 8 10 10000
```

Demo

Extreme Data

One Large Object




```
/simulate_n_body inputs/extreme_case_1.csv 8 10 10000
```

Implementation

Data Structure

Body Structure:

```
struct body_t {
  double mass;
  double position[3]; // x, y, z
  double velocity[3]; // x, y, z
  double net_force[3]; // x, y, z }
```

Force Matrix [N×N][3]

- Stores pairwise forces in vector (Fx, Fy, Fz)

Body_per_time[Total Time steps][N]

- Store updated body objects per time.

Simulation Parameters

- N (Number of bodies)
- Total Timesteps = Duration ÷ Step Size
- Example: 10s ÷ 0.1s = 100 iterations

Implementation

CUDA Parallel Processing Pipeline

1. Force Calculation

- N×N threads 1 block compute pairwise force
- index1 = threadIdx.x, index2 = threadIdx.y;
- Calculate force applied to index1 due to index2
  - **F_mag** = (G * m1 * m2)/(distance²) //r = distance
  - **F_x** = F_mag * (dx/distance) // Similarly for y, z

2. Net Force: Sum forces acting on each body

- N threads 1 block
- Each thread sums all forces acting on one body-> Store results in shared memory

3. Position & Velocity Update

- N threads 1 block
- Update all bodies in parallel
  - **v_new**= v_old + aΔt //(a = F/m)
  - **x_new**= x_new + v_oldΔt

Implementation

Output Generation

Writes simulation data to CSV

- **Records per-timestep data for each body:**
  - **Object ID, Mass, Px, Py, Pz, Vx, Vy, Vz, timestamp**

Visualization

- **Python matplotlib**

Thank you

Questions?

Project Available at : https://github.com/junghyey/csc-213-n-body

References

Chapter 31. Fast N-Body Simulation with CUDA. (n.d.). NVIDIA Developer. https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

Computation and astrophysics of the N-body problem. (n.d.). In Lecture 3 (p. 2). https://www.maths.ed.ac.uk/~heggie/lecture3.pdf

Wikipedia contributors. (2024, November 21). N-body problem.https://en.wikipedia.org/wiki/N-body_problem

https://pngtree.com/free-png-vectors/python-snake-clipart (python image)