The n-body problem in parallel

Damar Nicolás Rojas Chacón - 2122079
Diego Armando Villamizar Correa - 2161698
Miguel Angel Duarte Delgado - 2142665
Hordan Andrés Navarro - 2131799

02

SCALABILITY

04

RECOMMENDATIONS AND LIMITATIONS

01

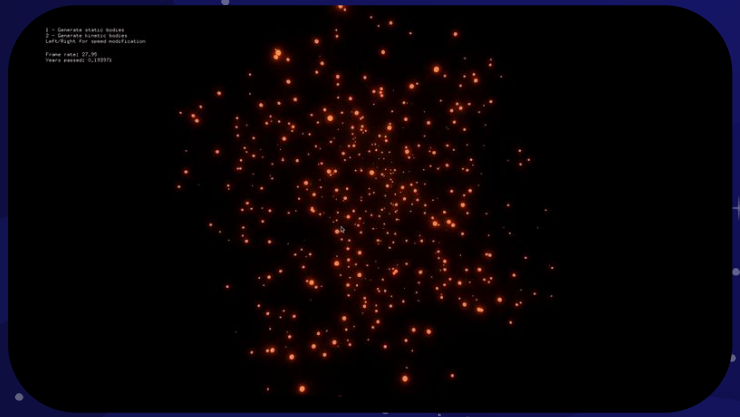
N-BODY PROBLEM

03

ACELERATION AND PROPOSED SOLUTION

01. N-BODY PROBLEM

In physics, the n-body problem is the problem of predicting the individual motions of a group of celestial objects interacting with each other gravitationally. Solving this problem has been motivated by the desire to understand the motions of: the Sun, Moon, planets, and visible stars.

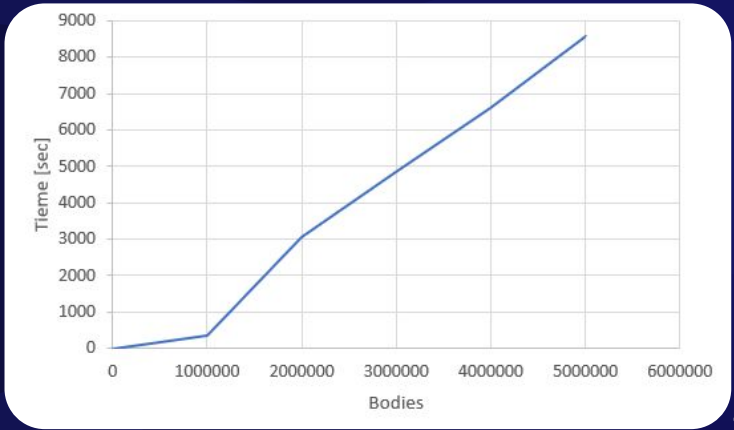


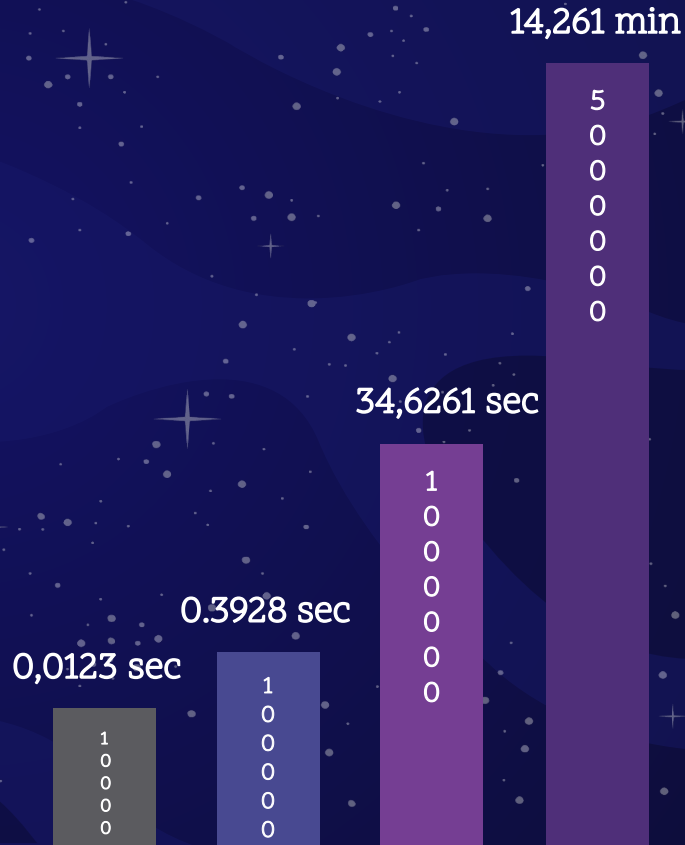
PROBLEM

Given the orbital properties (mass, instantaneous position and velocity) of a group of astronomical bodies, determine the interacting forces acting; and consequently, calculate their orbital motions for any future instant.

This for a minimum number of **100,000** bodies and preferably reaching more than **1’000.000**.

02. SCALABILITY





03. ACELERATION AND PROPOSED SOLUTION

1. Taking into account the source code, it was observed that the appropriate function to speed up was the bodyForce function.

2. The index is calculated for each thread to speed up the outermost for loop of this function.

3. We reserve space in the memory using CudaMalloc

4. We calculate the number of blocks and the number of threads will be 256 (multiple of 32).

5. We free the used space in memory using cudaFree

```
void bodyForce(float4 *p, float4 *v, float dt, int n) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < n) {
    float Fx = 0.0f; float Fy = 0.0f; float Fz = 0.0f;

    for (int tile = 0; tile < gridDim.x; tile++) {
      __shared__ float3 spos[BLOCK_SIZE];
      float4 tpos = p[tile * blockDim.x + threadIdx.x];
      spos[threadIdx.x] = make_float3(tpos.x, tpos.y, tpos.z);
      __syncthreads();

      for (int j = 0; j < BLOCK_SIZE; j++) {
        float dx = spos[j].x - p[i].x;
        float dy = spos[j].y - p[i].y;
        float dz = spos[j].z - p[i].z;
        float distSqr = dx*dx + dy*dy + dz*dz + SOFTENING;
        float invDist = rsqrtf(distSqr);
        float invDist3 = invDist * invDist * invDist;

        Fx += dx * invDist3; Fy += dy * invDist3; Fz += dz * invDist3;
      }
      __syncthreads();
    }

    v[i].x += dt*Fx; v[i].y += dt*Fy; v[i].z += dt*Fz;
  }
}
```

04. RECOMMENDATIONS AND LIMITATIONS

A strong limitation is that the algorithm is not scalable, so its performance does not grow with the increase of processors.

It is recommended to use a graphical tool that simulates the behavior of the n-bodies to better understand the problem and how they behave.

THanks!