Physics
of Data

MODERN COMPUTING FOR PHYSICS

N-BODIES SIMULATION WITH

GPU PARALLELIZATION

The n-bodies problem

- We consider a system of n **bodies**, each with a given **mass**, initial **position**, and initial **velocity**

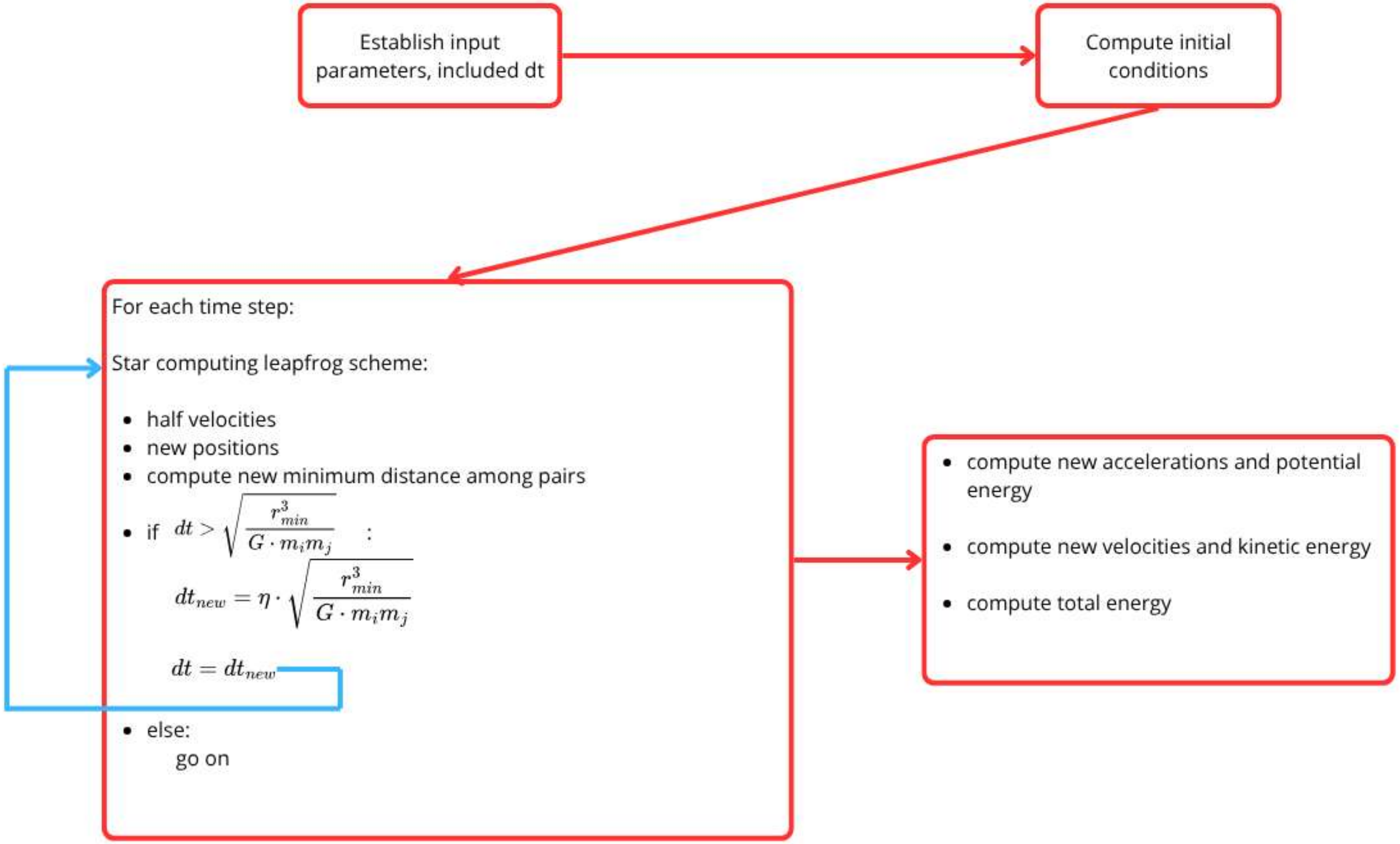
- The bodies interact solely through **gravitational forces**

- We want to simulate their **time evolution**, obtaining their trajectories

- We want to check the **conservation** of total **energy**

- Computational cost **O(n^2)**

Serial code in C



Establish input parameters, included dt

Compute initial conditions

For each time step:

Star computing leapfrog scheme:

- half velocities
- new positions
- compute new minimum distance among pairs
- if $dt > \sqrt{\frac{r_{min}^3}{G \cdot m_i m_j}}$:

  $dt_{new} = \eta \cdot \sqrt{\frac{r_{min}^3}{G \cdot m_i m_j}}$

  $dt = dt_{new}$
- else:

  go on

- compute new accelerations and potential energy
- compute new velocities and kinetic energy
- compute total energy

Issues: scale of the problem and data types

This scale is chosen such that:

-we can graphycally **observe trajectories:** gravitational forces daminatig over rectilinear uniform motion) and proportioned in the space

-the numerical scheme is **stable**

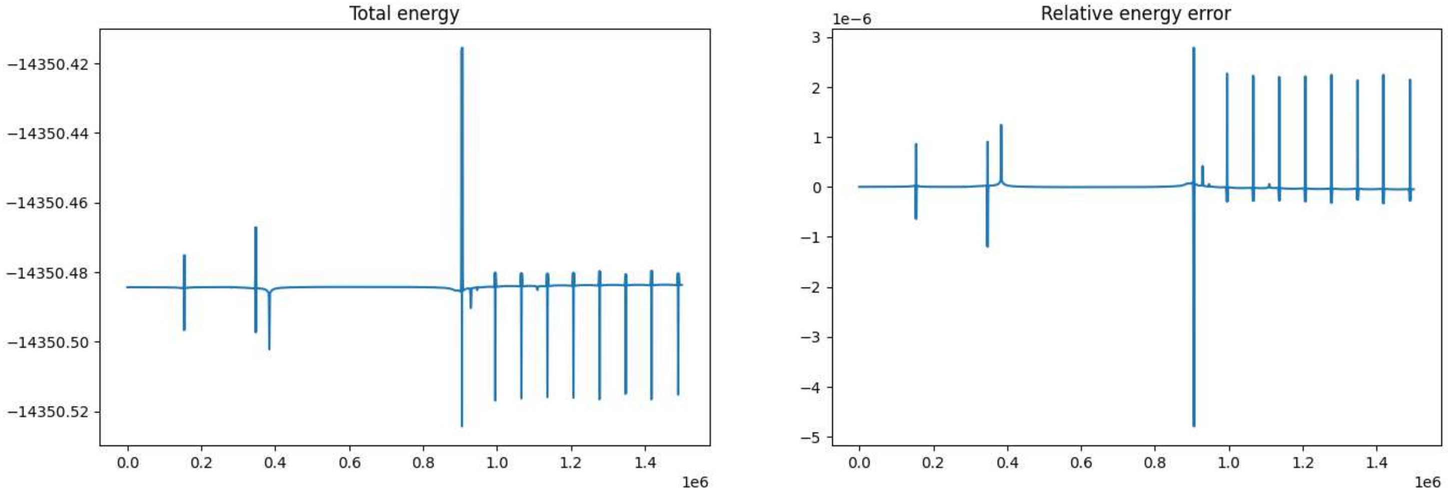
-you can compile with both float or **double** precision

```
for (int i = 0; i < n_bodies; i++) {
    m[i]  = rand_interval(1.0f, 10.0f);
    x[i]  = rand_interval(0.3f * L, 0.7f * L);
    y[i]  = rand_interval(0.3f * L, 0.7f * L);
    z[i]  = rand_interval(0.3f * L, 0.7f * L);
    vx[i] = rand_interval(-1.0f, 1.0f);
    vy[i] = rand_interval(-1.0f, 1.0f);
    vz[i] = rand_interval(-1.0f, 1.0f);
}
```

```
#ifdef FLOAT
typedef float real;
    #define SQRT  sqrtf
    #define FMOD fmodf
#else
    typedef double real;
    #define SQRT  sqrt
    #define FMOD fmod


#endif
```

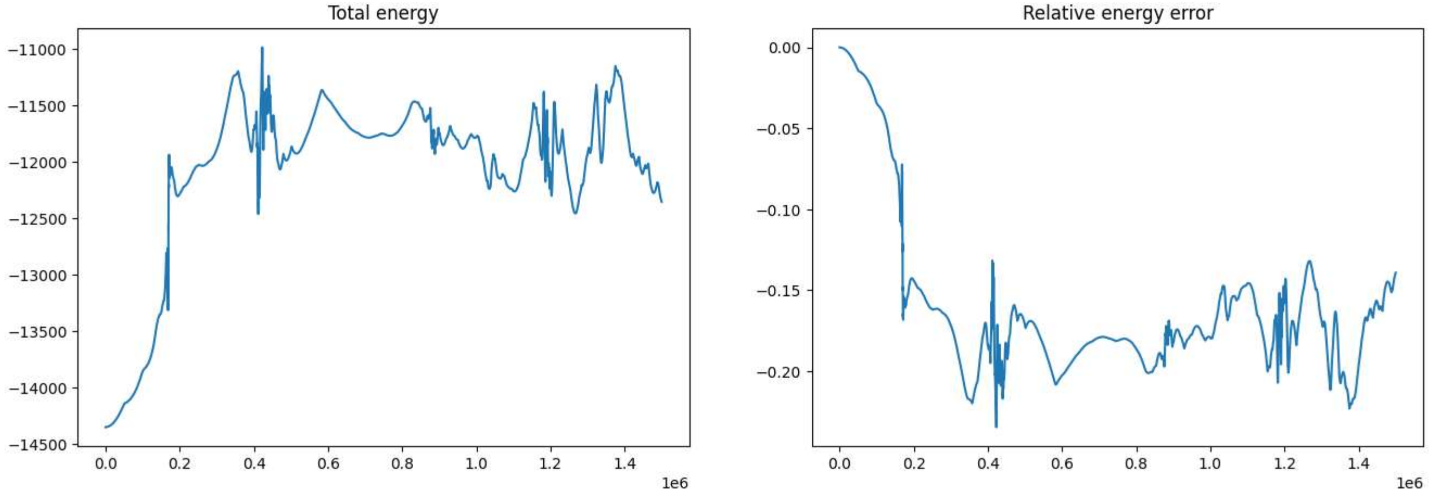
```
#define n_bodies 6
#define steps 1500000
#define SRAND 31265


const float dt_max = 1e-6f;
const float eta    = 1e-3f;
const float L      = 100.0f;
const float G      = 1.0e3f;
```

Evolution of total energy of the system



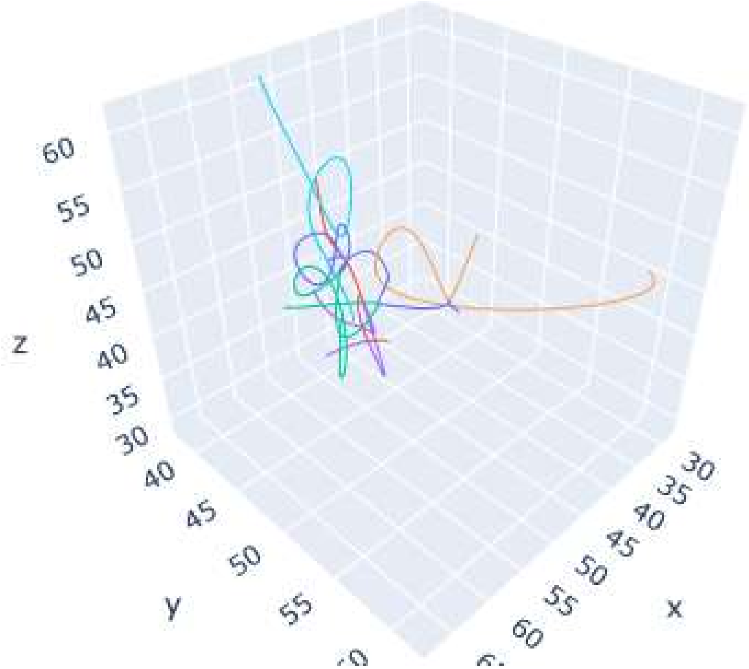
Double precision relative errors ~10^{-6}

Relative errors: $\dfrac{E_t}{E_{t=0}} - 1$



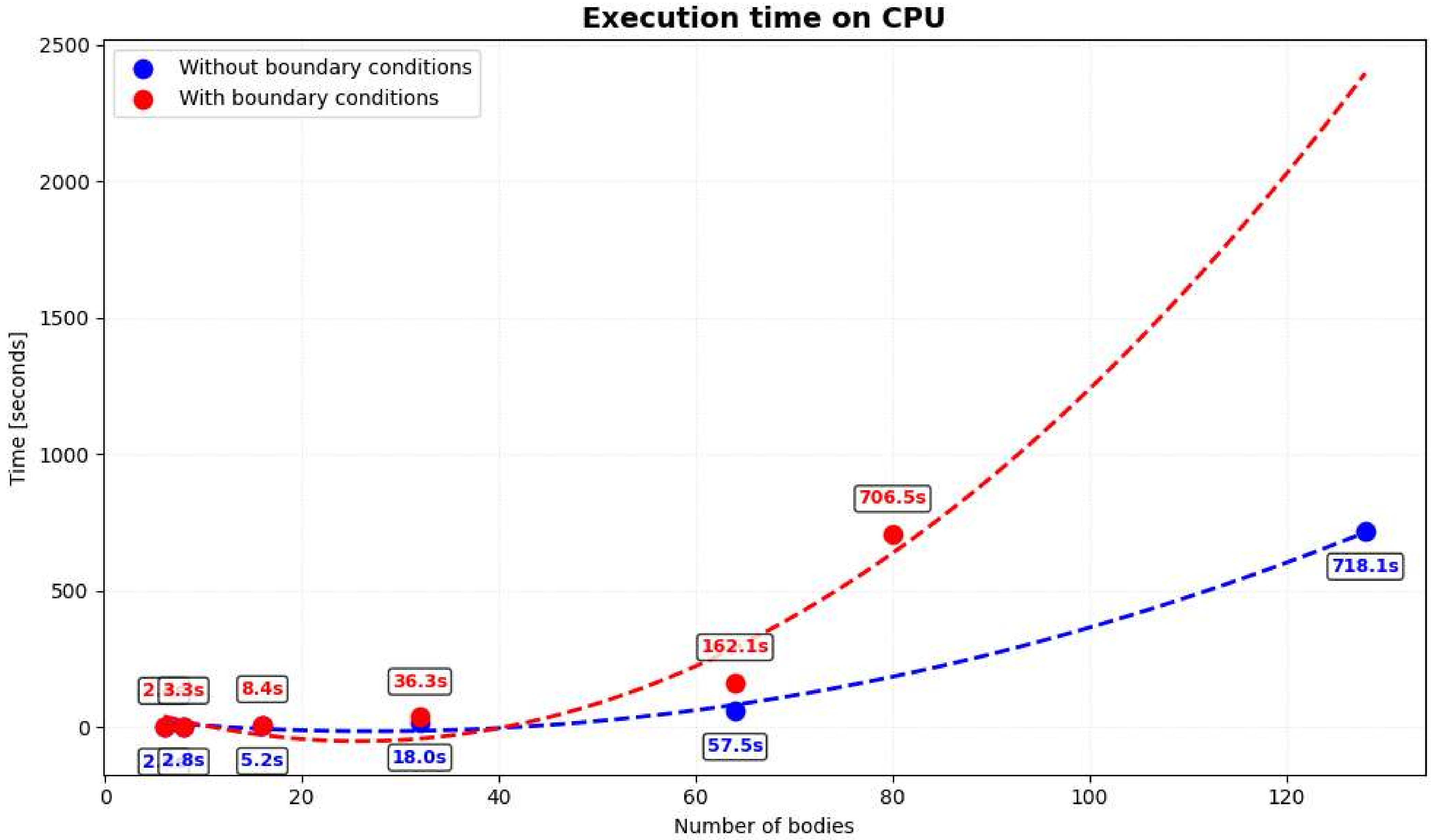
Float precision: unstable

6 bodies trajectories

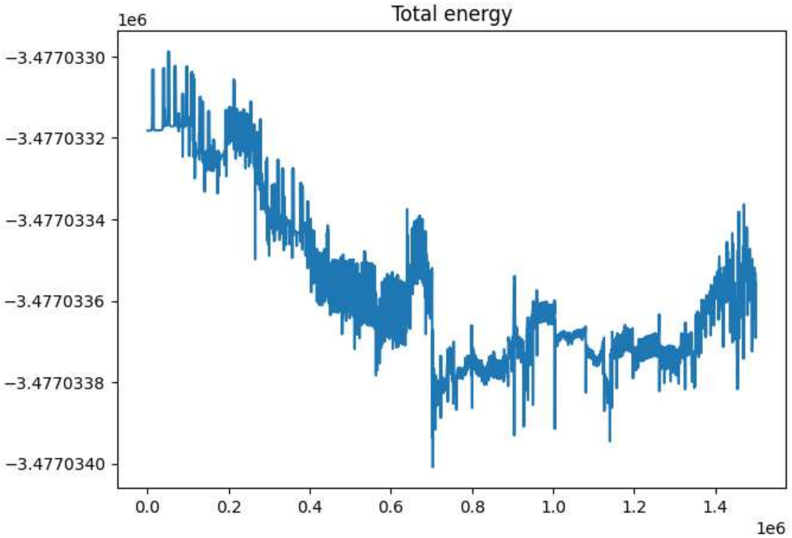


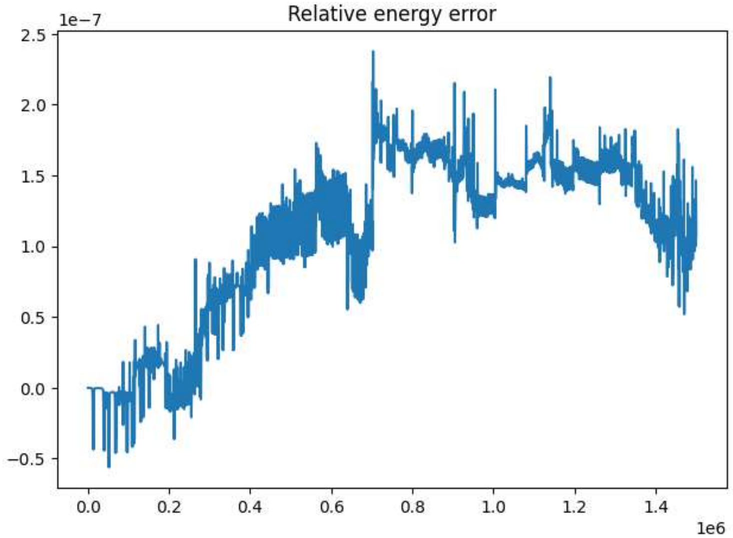
64 bodies (with periodic boundary conditions)

In the code a **csv** file to keep track of the positions has been created as output.
Trajectories have been plotten in python from this file with a script written by generative AI.
The csv has been modified through a script and be animated through *OVITO*.

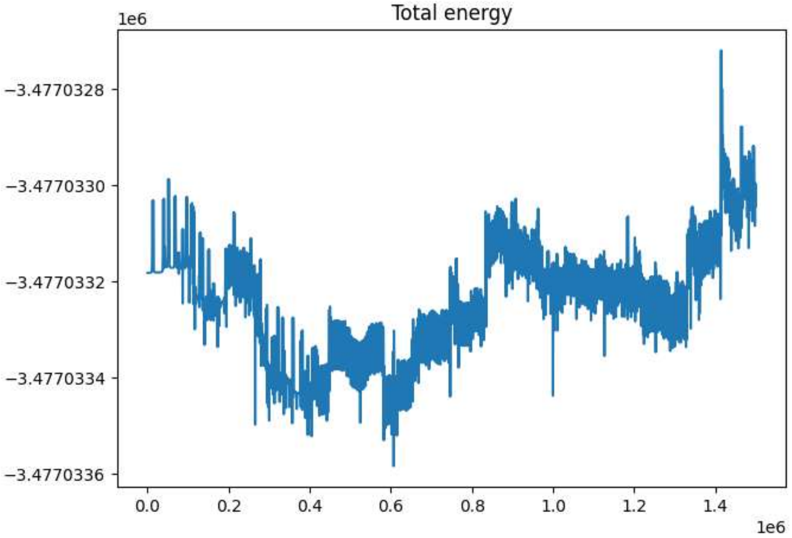

Execution time on CPU
Without boundary conditions
With boundary conditions
Time [seconds]
2500
2000
1500
1000
500
0
Number of bodies
0
20
40
60
80
100
120
2
3.3s
8.4s
36.3s
162.1s
706.5s
2
2.8s
5.2s
18.0s
57.5s
718.1s

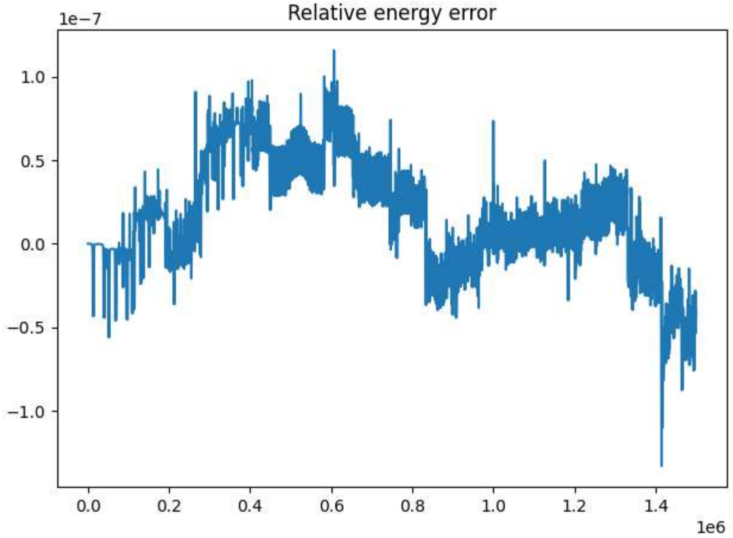
Conservation of energy for a 64 bodies system and 1500k steps





With periodic boundary conditions





Without boundary conditions

Parallel code in CUDA

GPU	Tesla T4
Global Memory	15360 MiB
Streaming MultiProcessors	40
Total CUDA Cores	2560
Max Threads per Block	1024
Shared Memory per SM	64 KB
Compute Capability	7.5

Crucial point is updating the accelerations of the bodies

⇒ **tiling** strategy is used, using the approach suggested in

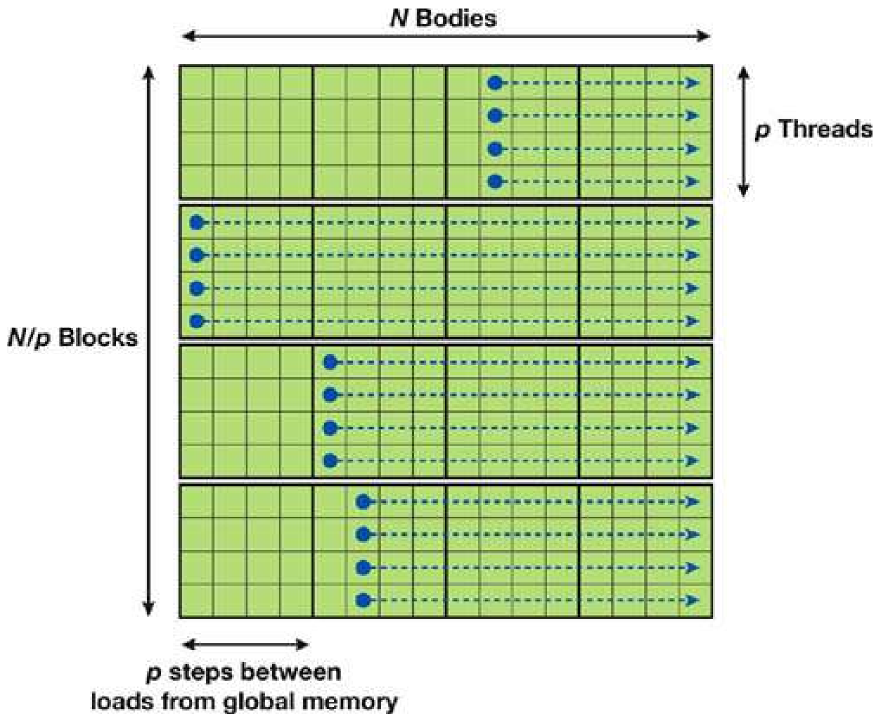
https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

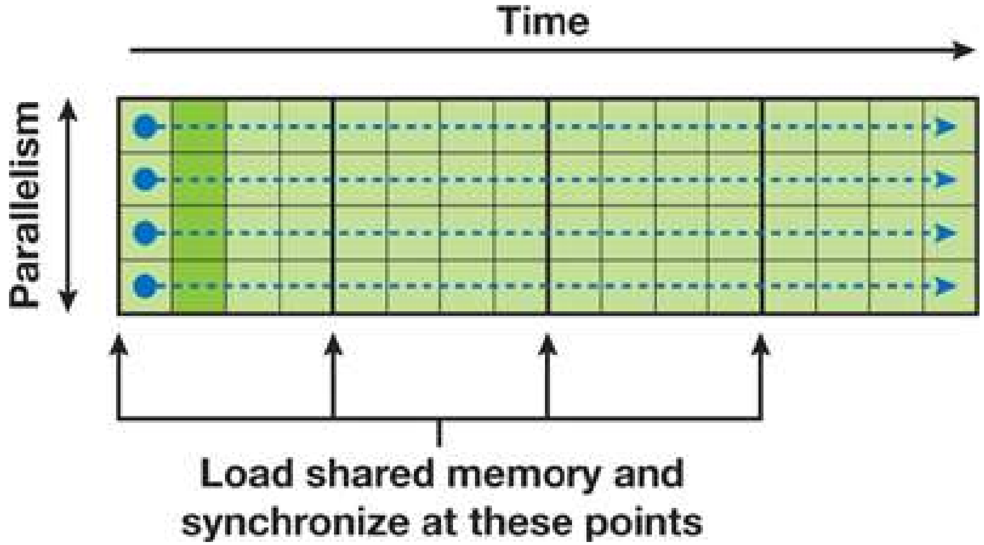
- We may think the algorithm as calculating each entry $\mathbf{a}_{ij}$ in an *N*x*N* grid of all pair-wise forces.
- Then the total acceleration $\mathbf{a}_i$ on body *i* is obtained from the sum of all entries in row *i*.

- O(N^2) would be too large

- We introduce tiles, square regions of the grid *pxp*

- Boundary conditions not applied





The properties of the bodies are written in shared memory of the blocks for each dimension of the grid.

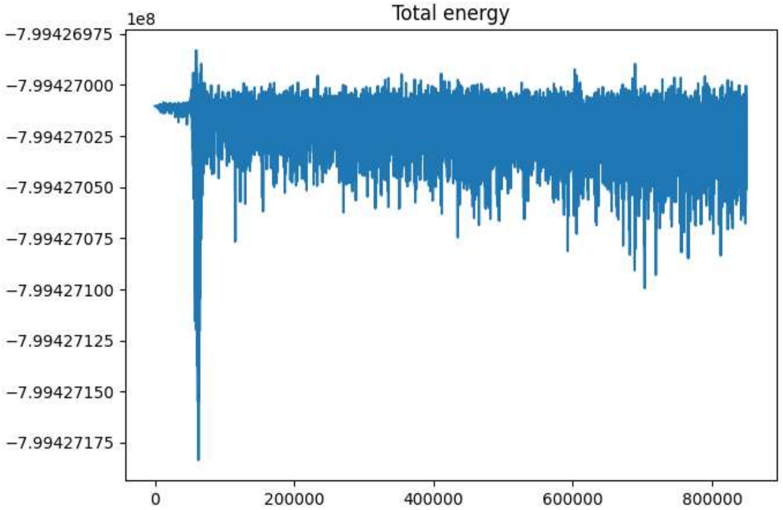
Each **row** corresponds to a **thread** which computes sequentially interactions with the bodies in the columns, summing up accelerations sequentially.

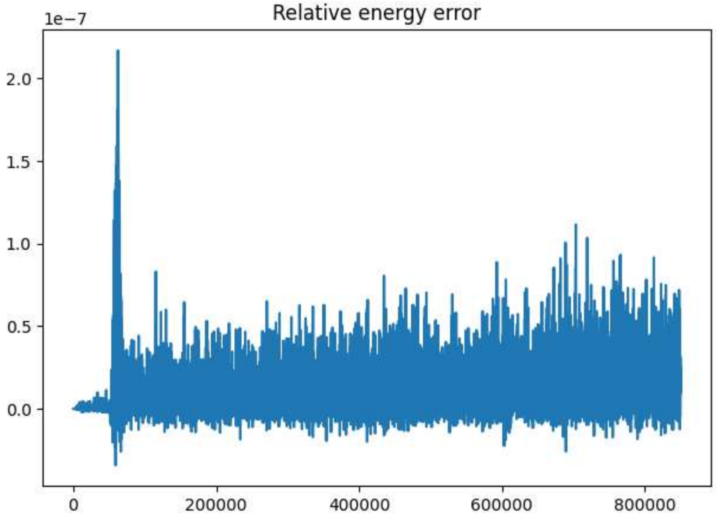
A final sum is computed among all tiles.

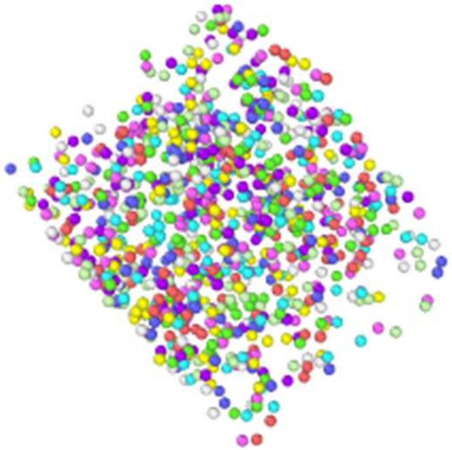
The problem's symmetry can’t be exploited, the real advantage is the **reduction of accesses to global memory**.

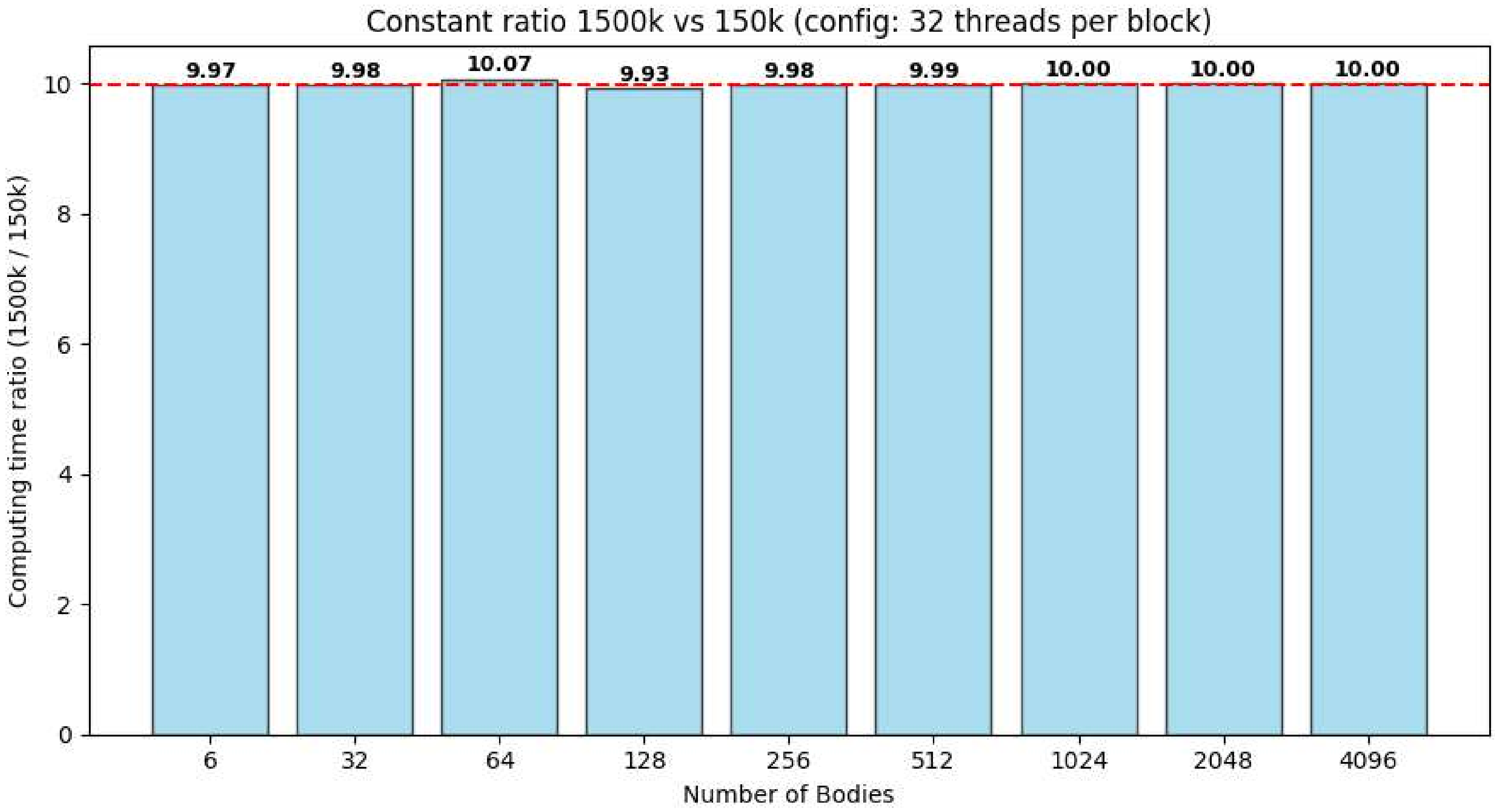
The algorithm preservs conservation of energy also with **1024** bodies

Relative errors ~10^{-7}

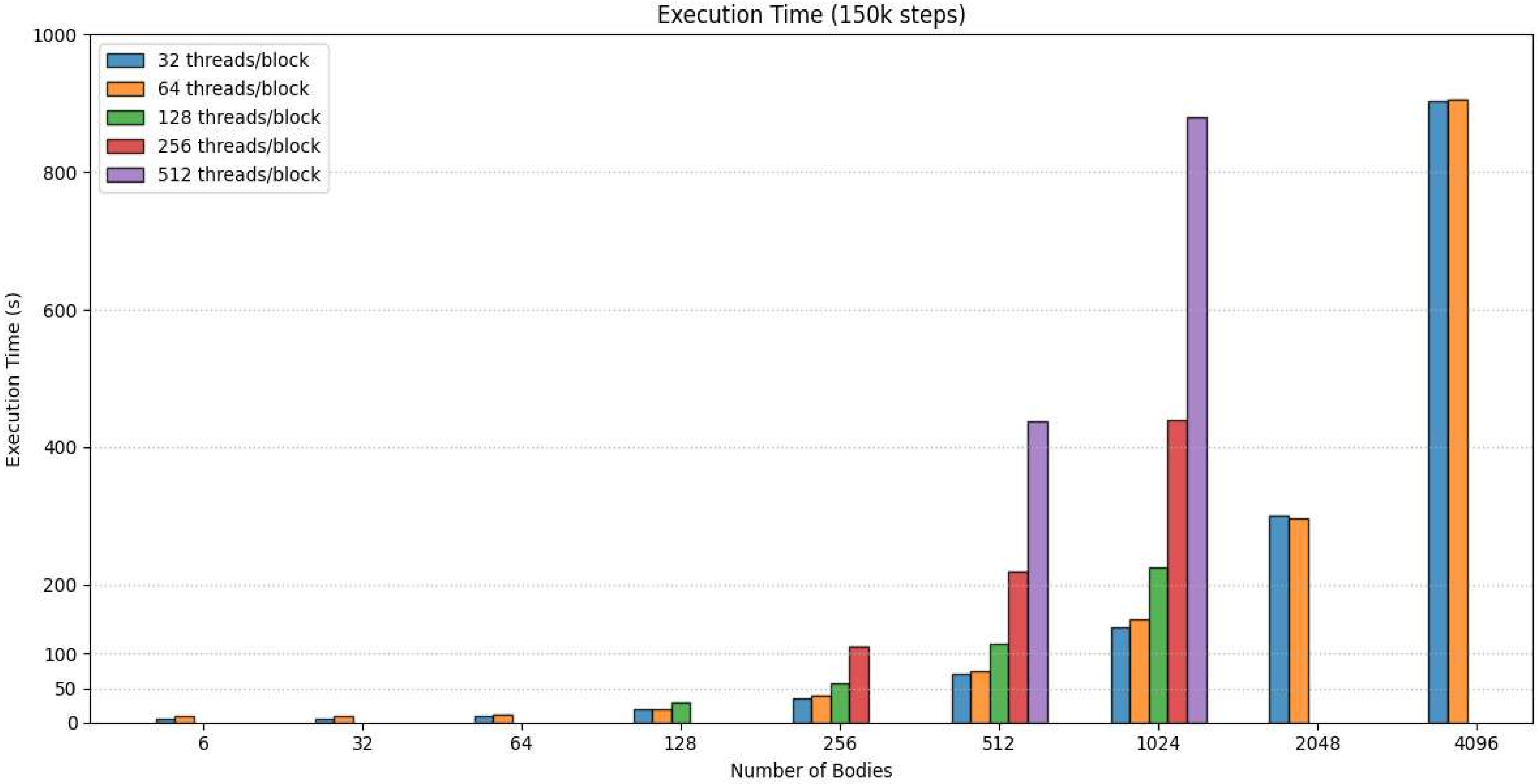






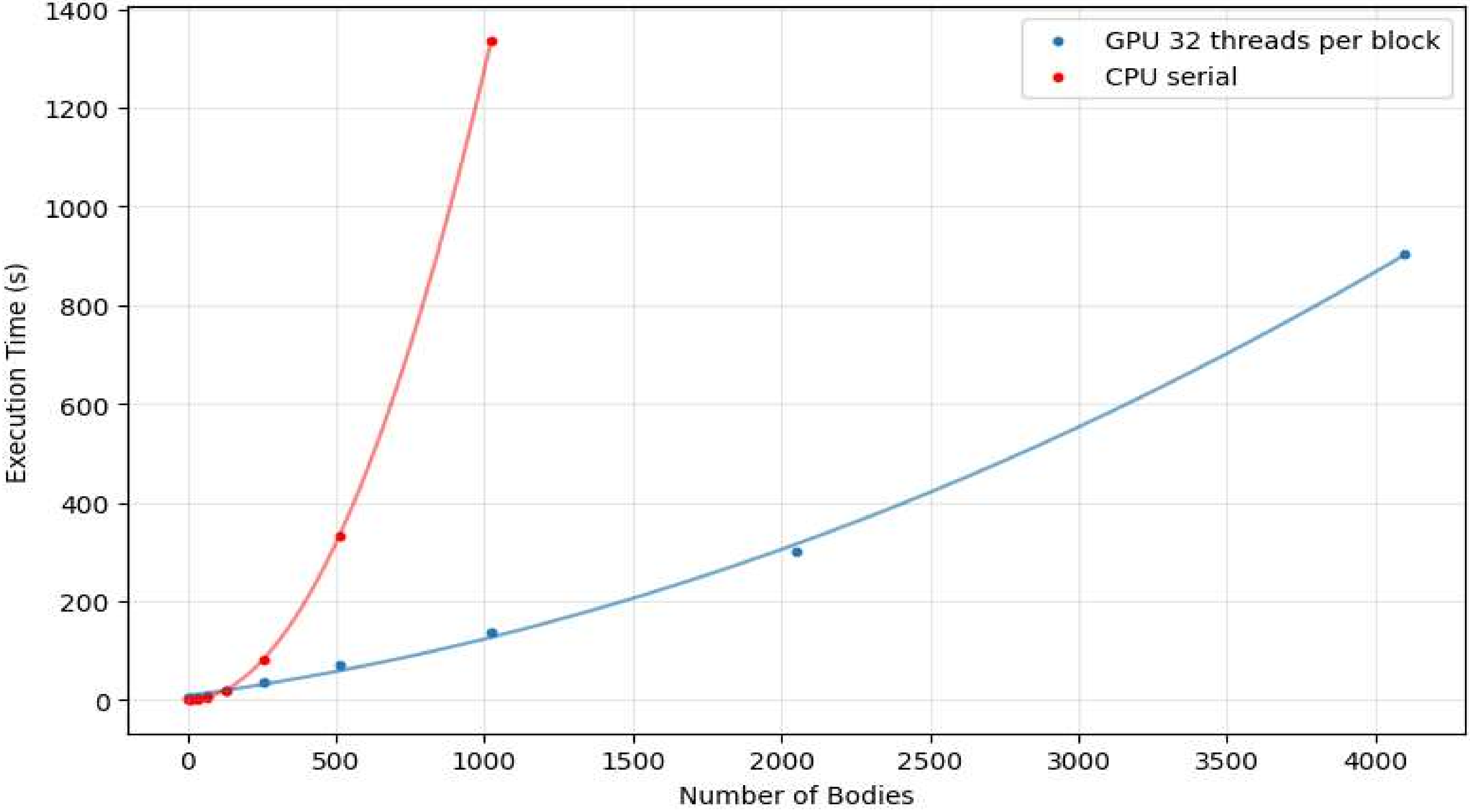


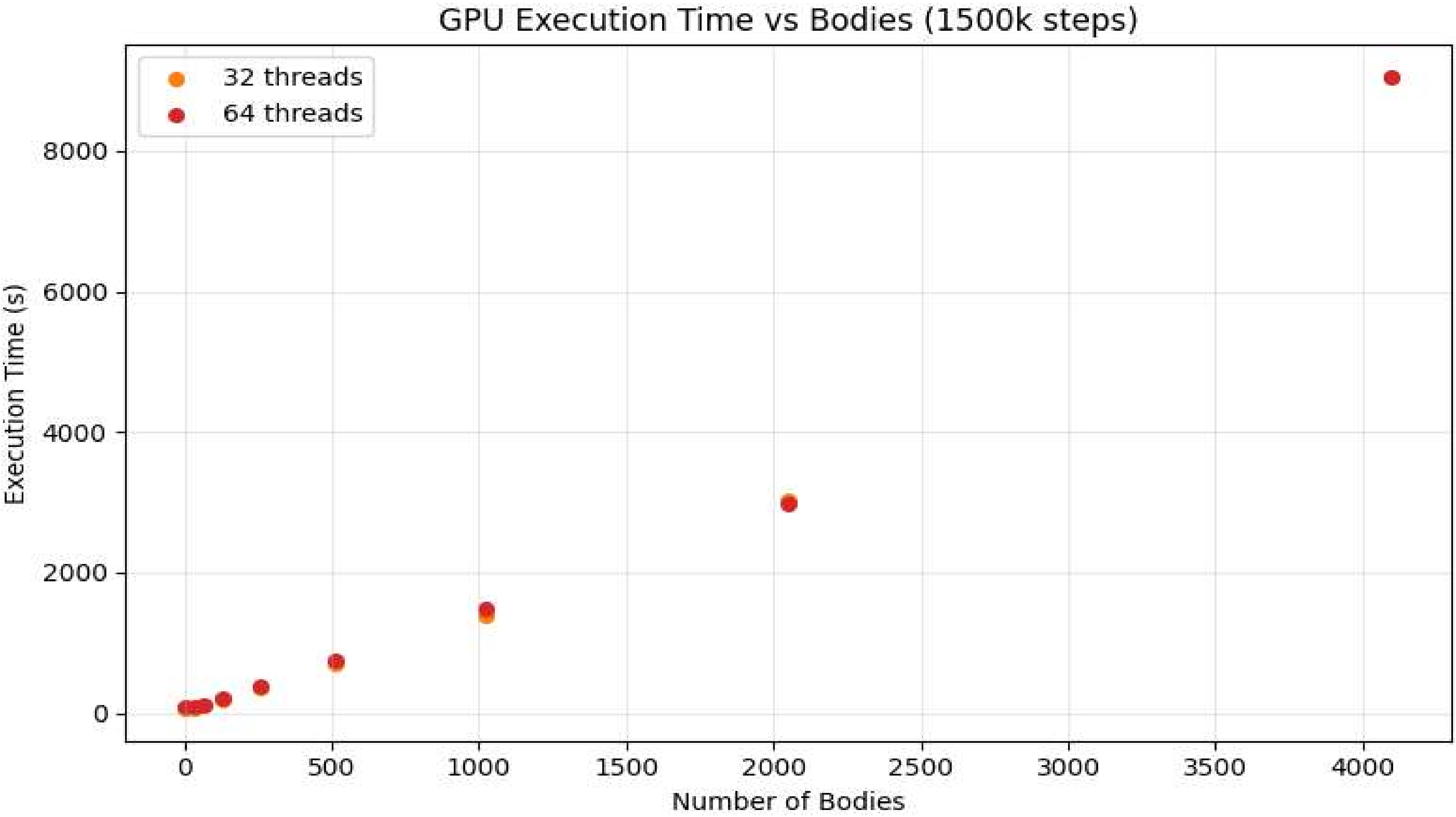
Execution time is linear with respect to the numer of steps.



n_blocks=n_bodies/thr_per_b

Having more blocks hides latency due to memory transfers.


GPU 32 threads per block
CPU serial
Execution Time (s)
Number of Bodies
0
200
400
600
800
1000
1200
1400
0
500
1000
1500
2000
2500
3000
3500
4000


GPU Execution Time vs Bodies (1500k steps)
32 threads
64 threads
Execution Time (s)
0
2000
4000
6000
8000
Number of Bodies
0
500
1000
1500
2000
2500
3000
3500
4000