Assignment 3: The Gravitational N-body Problem Report

Sotirios Oikonomou

February 11, 2026

1 Problem

The objective of this assignment is to implement a serial simulation of the two-dimensional gravitational N-body problem using the straightforward $O(N^2)$ algorithm[1, 4, 6]. The motion of N particles is governed by Newton's law of gravitation[1, 2] with Plummer softening [1, 4, 5] to avoid singular forces. Time integration is performed using the symplectic Euler method[1, 3].

The simulation reads the initial particle configuration from a binary file, and evolves the system for a specified number of timesteps, and writes the final configuration back to a file in the same format. The gravitational constant is defined as

$$G = \frac{100}{N}, \tag{1}$$

and the timestep size Δt and softening parameter $\varepsilon_0 = 10^{-3}$ [1] are provided as input parameters.

The main computational task is the evaluation of pairwise interactions between particles, leading to quadratic time complexity per timestep [4, 6].

2 Solution

2.1 Data Representation

Each particle is stored as six double values in the required binary format:

$(x, y, m, vx, vy, brightness)$.

The brightness value is not used in the simulation, but is preserved in the output to maintain the required file format.

The particle data is stored in a single dynamically allocated contiguous array:

```
double * data = malloc (6* N * sizeof ( double ) ) ;
```

The layout for particle i is accessed as:

- data[6*i + 0] → x
- data[6*i + 1] → y
- data[6*i + 2] → m
- data[6*i + 3] → vx
- data[6*i + 4] → vy
- data[6*i + 5] → brightness

This contiguous layout provides good spatial locality and is suitable for performance work later.

2.2 Force Computation

For each timestep, the force (and acceleration) on each particle is computed using a double loop over all pairs (i, j) with $j \neq i$. With Plummer softening, the force on particle i is

$$\mathbf{F}_i = -Gm_i \sum_{j \neq i} m_j \frac{\mathbf{r}_{ij}}{(r_{ij} + \varepsilon_0)^3}, \tag{2}$$

where $\mathbf{r}_{ij} = (x_i - x_j, y_i - y_j)$ and $r_{ij} = \|\mathbf{r}_{ij}\|$.

The acceleration is $\mathbf{a}_i = \mathbf{F}_i / m_i$, so the mass m_i cancels:

$$\mathbf{a}_i = -G \sum_{j \neq i} m_j \frac{\mathbf{r}_{ij}}{(r_{ij} + \varepsilon_0)^3}. \tag{3}$$

The implementation uses two nested loops:

```
for ( int i = 0; i < N ; i ++) {
    for ( int j = 0; j < N ; j ++) {
        if ( j == i ) continue ;
        ...
    }
}
```

This yields $O(N^2)$ work per timestep [1, 4, 5].

To avoid mixing updated and non-updated particle states during force computation, temporary arrays ax and ay store the acceleration components for all particles before any update is applied .

2.3 Time Integration

Time stepping is done using the symplectic Euler method:

$$v^{n+1} = v^n + \Delta t \, a^n, \tag{4}$$

$$x^{n+1} = x^n + \Delta t \, v^{n+1}. \tag{5}$$

The key property is that the position update uses the updated velocity v^{n+1}.

Each timestep is implemented in two phases:

1. Compute accelerations for all particles using the positions at time level n.
2. Update velocities, then positions for all particles (symplectic Euler).

The timestep loop is structured as:

```
for ( int step = 0; step < nsteps ; step ++) {
    // compute accelerations ax [ i ] , ay [ i ]
    // update velocities
    // update positions
}
```

3 Verification of Correctness

The implementation was validated using the provided reference output files. For the test case:

- $N = 10$
- input: ellipse_N_00010.gal
- timesteps: 200
- $\Delta t = 10^{-5}$

The produced output result.gal was compared against ellipse_N_00010_after200steps.gal using the supplied compare_gal_files program [1]. The maximum positional difference reported was:

```
pos_maxdiff = 0.0,
```

confirming that the implementation reproduces the reference result exactly for this case.

In addition, small test inputs (e.g., circles_N_2.gal) were used to sanity-check the timestep update, confirming that particle positions and velocities change consistently with the computed gravitational forces.

4 Performance and Discussion

4.1 Experimental Setup

Performance measurements were conducted using the shell `time` command. The reported values correspond to the `user` CPU time, which measures the actual processor time spent executing the program.

All experiments were performed with:

- Number of timesteps: 100
- Timestep size: $\Delta t = 10^{-5}$
- Graphics disabled

For each value of N, the program was executed five times under identical conditions. The execution times reported in the tables below correspond to the arithmetic mean of these five runs. Averaging reduces the influence of operating system scheduling, background processes, and measurement variability.

4.2 Measured Execution Times

The following average execution times were obtained using compiler optimization level `-O2`.

N	Average time (s)
500	0.040
1000	0.146
2000	0.590
3000	1.340

Table 1: Average execution time using `-O2`.

The following average execution times were obtained using compiler optimization level `-O3 -march=native`.

The case $N = 100$ was excluded from the scaling analysis because the execution time was too small to measure reliably at the resolution of the timing tool.

N	Average time (s)
500	0.040
1000	0.146
2000	0.592
3000	1.316

Table 2: Average execution time using `-O3 -march=native`.

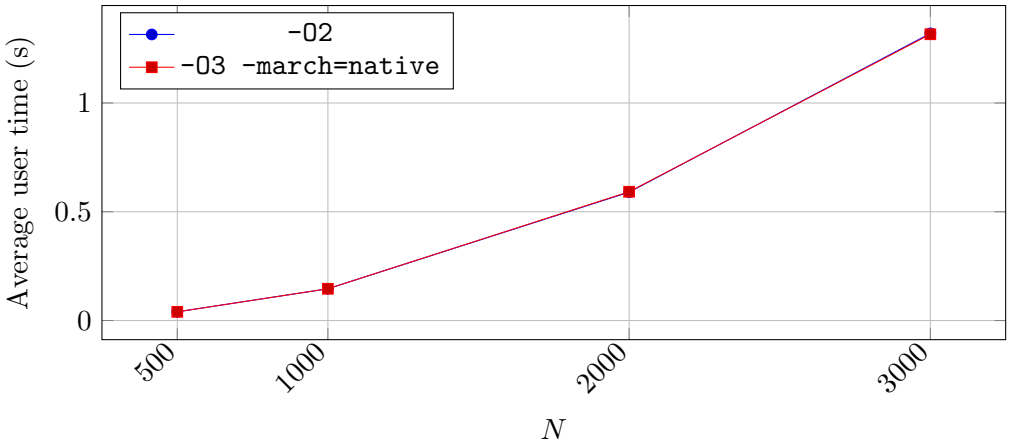

Figure 1: Execution time as a function of N for 100 timesteps and $\Delta t = 10^{-5}$ (graphics off). Each point is the arithmetic mean of 5 runs.

4.3 Scaling Behaviour

The straightforward N-body algorithm performs pairwise interactions between all particles, leading to $\mathcal{O}(N^2)$ computational complexity per timestep [4, 6].

To verify this behaviour experimentally, we examine how runtime changes when doubling N.

From $N = 500$ to $N = 1000$:

$$\frac{0.146}{0.040} \approx 3.65$$

From $N = 1000$ to $N = 2000$:

$$\frac{0.592}{0.146} \approx 4.05$$

Ideally, doubling N in a quadratic algorithm should increase runtime by a factor of four. The measured ratios are very close to this theoretical value, confirming $\mathcal{O}(N^2)$ scaling.

Furthermore, increasing from $N = 1000$ to $N = 3000$ should increase runtime by a factor of $3^2 = 9$. The measured ratio is:

$$\frac{1.316}{0.146} \approx 9.01,$$

which closely matches the expected behaviour.

These results demonstrate that the implementation follows the theoretical quadratic complexity very accurately.

4.4 Effect of Compiler Optimizations

Two compiler configurations were evaluated:

- `-O2`

- `-O3 -march=native`

The performance difference between the two configurations was negligible. This indicates that the compiler already applies sufficient loop optimizations and instruction scheduling at `-O2`, and additional aggressive optimization flags provide limited additional benefit for this implementation.

4.5 Manual Optimization: Symmetry / Newton's 3rd Law

After establishing the baseline $O(N^2)$ implementation, a manual optimization was added to reduce redundant force computations. Instead of computing all ordered pairs (i, j) and skipping $i = j$, we compute only pairs with $j > i$ and update the accelerations of both particles from the same interaction (Newton's third law). This almost halves the number of pair interactions, while preserving the same physical model and numerical method [2, 4].

4.6 Measured Execution Times After Optimization

The following average execution times were obtained with the symmetry optimization enabled, using `-O2`.

N	Average time (s)
500	0.040
1000	0.120
2000	0.452
3000	1.188

Table 3: Average execution time after symmetry optimization using `-O2`.

The following average execution times were obtained with the symmetry optimization enabled, using `-O3 -march=native`.

N	Average time (s)
500	0.038
1000	0.122
2000	0.454
3000	1.140

Table 4: Average execution time after symmetry optimization using `-O3 -march=native`.

4.7 Speedup From the Optimization

To quantify the effect, we compute the speedup as

$$S = \frac{T_{\text{baseline}}}{T_{\text{optimized}}}.$$

4.8 Execution Time Plot Including Optimized Code

	`-O2`			`-O3 -march=native`		
N	Baseline (s)	Opt (s)	Speedup	Baseline (s)	Opt (s)	Speedup
500	0.040	0.040	1.00	0.040	0.038	1.05
1000	0.146	0.120	1.22	0.146	0.122	1.20
2000	0.590	0.452	1.31	0.592	0.454	1.30
3000	1.340	1.188	1.13	1.316	1.140	1.15

Table 5: Speedup achieved by the symmetry optimization compared to the baseline implementation.

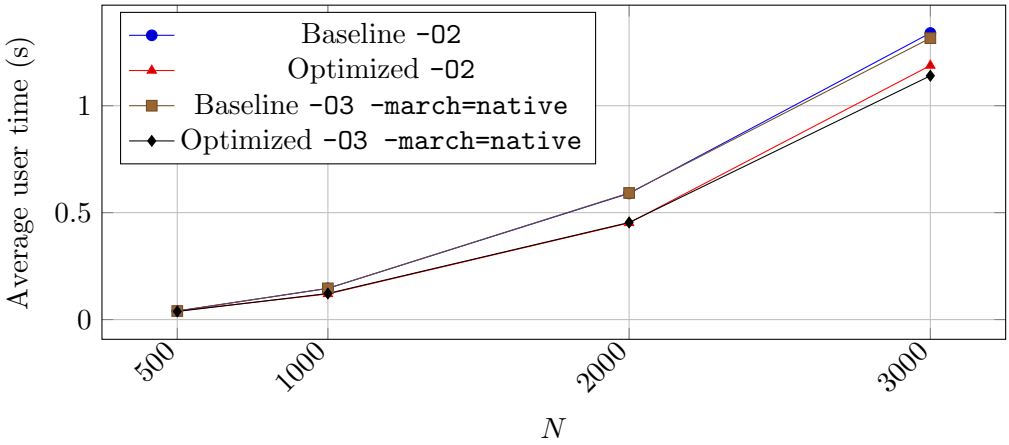

Figure 2: Baseline vs. optimized execution time as a function of N for 100 timesteps and $\Delta t = 10^{-5}$ (graphics off).

5 Reproducibility

All experiments were performed with graphics disabled and with fixed simulation parameters: 100 timesteps and $\Delta t = 10^{-5}$. Each timing configuration was executed five times and the reported values correspond to the arithmetic mean of the five `user` CPU times measured using the shell `time` command.

The fastest time reported in `best_timing.txt` is reported in *wall seconds* (elapsed time), and was obtained from the `total/real` time reported by `time`.

System information. CPU model: `Apple M4 Pro`
Operating system: `ProductName: macOS, ProductVersion: 26.2, BuildVersion: 25C56`
Compiler: `Apple clang version 17.0.0 (clang-1700.6.3.2)`

Compilation. The program was compiled using:

- `gcc -O2 -Wall -Wextra galaxy.c -o galsim -lm`
- `gcc -O3 -march=native -Wall -Wextra galaxy.c -o galsim -lm`

Benchmark command. All benchmarks were run using:

```
./galsim N input_data/ellipse_N_XXXXX.gal 100 1e-5 0
```

where $N \in \{500, 1000, 2000, 3000\}$ and the input file matched the value of N.

References

[1] High Performance Programming, Uppsala University, *Assignment 3: The Gravitational N-body Problem*, Spring 2022.

[2] I. Newton, *Philosophiæ Naturalis Principia Mathematica*, 1687.

[3] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer Series in Computational Mathematics, 2nd ed., Springer, 2006.

[4] S. J. Aarseth, *Gravitational N-Body Simulations: Tools and Algorithms*, Cambridge University Press, 2003.

[5] H. C. Plummer, "On the problem of distribution in globular star clusters," *Monthly Notices of the Royal Astronomical Society*, vol. 71, pp. 460–470, 1911.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.