Assignment 3

Ting-Hsuan Lien

Problem

Calculate the evolution of N particles in a gravitational simulation.

Solution

Data structure (for particle)

Simply record all information of particle

```
/* Structure to store the state of a particle
 * x: position x
 * y: position y
 * mass: mass of the particle
 * v_x: velocity x
 * v_y: velocity y
 * brightness: brightness of the particle
 */
typedef struct
{
    double x, y, mass, v_x, v_y, brightness;
} Particle;
```

Structure (code)

My code can spilt into 6 main part.

Particle: Contains the definition of the `Particle` structure and function `Print()` for debugging purposes.

ParseArguments: reading input data from files to initialize particle properties.

ReadFile: read input data and create particles.

Simulation: The core of the program where all computations occur.

WriteFile: writing final result in to result.gal

FreeMemory: release all memory in the end.

Algorithm

Algorithm implement in Simulation part. Following are pseudocode.

```
for t in nsteps:
    for i in N:
        F = 0.0;
        for j in N:
            if i != j:
                dx = particles[i]->x - particles[j]->x;
                dy = particles[i]->y - particles[j]->y;
                r = sqrt((dx * dx) + (dy * dy));
                F += particles[j]->mass / (r * r * r);
        F *= -G * particles[i]->mass;
        a = F / particles[i]->mass;
        update paricles[i] velocity
    for i in N:
        update paricles[i] position
```

Performance and Discussion

All experiments run same data in 200 step (which has ref_output).

Original Version (without any optimizations)

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.002s	0m0.000s
ellipse_N_00100	0m0.027s	0m0.020s	0m0.000s
ellipse_N_00500	0m0.485s	0m0.456s	0m0.000s
ellipse_N_01000	0m1.890s	0m1.798s	0m0.000s

INPUT DATA	REAL	USER	SYS
ellipse_N_02000	0m6.200s	0m7.277s	0m0.000s

In the **Original Version**, no specific optimizations were applied. The graph below clearly shows an **$O(n^2)$** curve.

Figure 1: original version - real time performance

Compile with `-o3`

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.015s	0m0.008s	0m0.000s
ellipse_N_00500	0m0.179s	0m0.163s	0m0.000s
ellipse_N_01000	0m0.681s	0m0.642s	0m0.000s
ellipse_N_02000	0m1.261s	0m2.587s	0m0.000s

In this version, no modifications were made to the code; only the `-o3` optimization flag was used during compilation. The execution time was significantly reduced. In the largest test case, the **real time** decreased by **79.6%**.

In Figure 2 the time complexity seem not like **$O(n^2)$**, but no modification of code, the algorithm still **$O(n^2)$**.

Figure 2: Compile with `-o3` version - real time performance

Compile with `-funroll-loops`

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.009s	0m0.002s	0m0.000s
ellipse_N_00100	0m0.016s	0m0.008s	0m0.000s
ellipse_N_00500	0m0.180s	0m0.164s	0m0.000s

INPUT DATA	REAL	USER	SYS
ellipse_N_01000	0m0.688s	0m0.648s	0m0.000s
ellipse_N_02000	0m1.271s	0m2.602s	0m0.001s

In this version, same as last version, just add another compile flag. It didn't effect the runtime (< 0.01s).

Move `F()` into main

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.002s	0m0.000s
ellipse_N_00100	0m0.015s	0m0.008s	0m0.000s
ellipse_N_00500	0m0.177s	0m0.152s	0m0.009s
ellipse_N_01000	0m0.671s	0m0.631s	0m0.000s
ellipse_N_02000	0m1.169s	0m2.516s	0m0.010s

Initially, I placed the code in `F()` for easy to modify. In this version, I moved it into `main` function to reduce function call. In the larger test case (N = 2000), this optimization reduced the execution time by 0.1s compared to before.

Figure 3 looks similar to Figure 2. but the peak is slightly lower.

Figure 3: Move `F()` into main - real time performance

Remove if (i != j) in loop

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.011s	0m0.005s	0m0.000s
ellipse_N_00500	0m0.103s	0m0.090s	0m0.000s
ellipse_N_01000	0m0.377s	0m0.342s	0m0.009s
ellipse_N_02000	0m1.487s	0m1.406s	0m0.000s

As shown in the pseudocode in *Algorithm*, the `for` loop originally used `if (i != j)` to skip the calculation for the particle itself. Since conditional checks inside loops can slow down execution, I modified the code to avoid using `if` statements inside the loop.

The new version pseudocode is following:

```
for t in nsteps:
    for i in N:
        F = 0.0;
        for j in range(0, i):
            dx = particles[i]->x - particles[j]->x;
            dy = particles[i]->y - particles[j]->y;
            r = sqrt((dx * dx) + (dy * dy));
            F += particles[j]->mass / (r * r * r);
        for j in range(i+1, N):
            dx = particles[i]->x - particles[j]->x;
            dy = particles[i]->y - particles[j]->y;
            r = sqrt((dx * dx) + (dy * dy));
            F += particles[j]->mass / (r * r * r);
        F *= -G * particles[i]->mass;
        a = F / particles[i]->mass;
        update paricles[i] velocity
    for i in N:
        update paricles[i] position
```

In **Figure 4**, the runtime for **"ellipse_N_01000"** shows a significant reduction, but the other cases did not change much. Notably, the runtime for "ellipse_N_02000" even increased.

Figure 4: Remove if (i != j) in loop - real time performance

Change for to do-while

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.012s	0m0.005s	0m0.000s
ellipse_N_00500	0m0.102s	0m0.090s	0m0.000s

INPUT DATA	REAL	USER	SYS
ellipse_N_01000	0m0.383s	0m0.357s	0m0.000s
ellipse_N_02000	0m1.499s	0m1.408s	0m0.010s

In this version, I changed most of the loops from `for` to `do-while`, but it did not improve performance.

Move `Destroy()` into main

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.011s	0m0.005s	0m0.000s
ellipse_N_00500	0m0.098s	0m0.086s	0m0.000s
ellipse_N_01000	0m0.363s	0m0.336s	0m0.000s
ellipse_N_02000	0m1.407s	0m1.328s	0m0.000s

Similar to *Move `F()` into main*, I moved `Destroy()` to `main`. It had a slight positive effect.

Reduce useless calculation

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.008s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.011s	0m0.005s	0m0.000s
ellipse_N_00500	0m0.098s	0m0.081s	0m0.009s
ellipse_N_01000	0m0.356s	0m0.335s	0m0.010s
ellipse_N_02000	0m1.428s	0m1.410s	0m0.000s

Lastly, I combined some redundant calculations and used faster operators. However, the effect was still not significant.

After modification, the new code is following:

```
for t in nsteps:
    for i in N:
        F = 0.0;
        for j in range(0, i):
            dx = particles[i]->x - particles[j]->x;
            dy = particles[i]->y - particles[j]->y;
            r = sqrt((dx * dx) + (dy * dy));
            F += particles[j]->mass / (r * r * r);
        for j in range(i+1, N):
            dx = particles[i]->x - particles[j]->x;
            dy = particles[i]->y - particles[j]->y;
            r = sqrt((dx * dx) + (dy * dy));
            F += particles[j]->mass / (r * r * r);
        particles[i]->v += - G * F * delta_t;
    for i in N:
        update paricles[i] position
```

Modify Array

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.007s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.010s	0m0.004s	0m0.000s
ellipse_N_00500	0m0.095s	0m0.090s	0m0.000s
ellipse_N_01000	0m0.341s	0m0.340s	0m0.000s
ellipse_N_02000	0m1.328s	0m1.351s	0m0.000s
ellipse_N_03000	0m1.494s	0m1.520s	0m0.000s

```
typedef struct
{
    double* x;
    double* y;
    double* mass;
    double* v_x;
    double* v_y;
    double* brightness;
} Particle;
```

Modify the structure for better vectorization. The execution time was significantly reduced, especially in the case of "**ellipse_N_03000**", where the **real time** decreased from **0m3.118s** to **0m1.494s**.

Reduce the number of computations

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.007s	0m0.001s	0m0.000s
ellipse_N_00100	0m0.011s	0m0.005s	0m0.000s
ellipse_N_00500	0m0.104s	0m0.100s	0m0.000s
ellipse_N_01000	0m0.388s	0m0.386s	0m0.010s
ellipse_N_02000	0m1.527s	0m1.578s	0m0.000s
ellipse_N_03000	0m1.703s	0m1.769s	0m0.000s

Compared to the previous version, where velocity and position updates were performed in a separate loop after force calculations, this version integrates the updates directly within the force computation loop.

The execution time slightly increased, especially for larger cases (**N = 2000, 3000**).

Vectorized

INPUT DATA	REAL	USER	SYS
ellipse_N_00010	0m0.009s	0m0.002s	0m0.000s
ellipse_N_00100	0m0.007s	0m0.002s	0m0.000s
ellipse_N_00500	0m0.031s	0m0.025s	0m0.001s
ellipse_N_01000	0m0.095s	0m0.090s	0m0.000s
ellipse_N_02000	0m0.340s	0m0.353s	0m0.000s
ellipse_N_03000	0m0.387s	0m0.391s	0m0.011s

By introducing additional variables, modifications within the **for j** loop no longer directly affect `F_x[i]` and `F_y[i]`, allowing for **vectorization**. After **vectorization**, the execution time was significantly reduced, especially for larger test cases.

Discussion

In conclusion, the most effective optimization was clearly compiling with `-O3`. Also structure effect a lot. Reducing conditional statements inside loops also had a noticeable impact. Reducing function calls showed some improvement in larger test cases. Other optimization strategies did not produce significant effects in this assignment.

Environment

Run on Windows 11 wsl.

CPU: AMD Ryzen 5 7535HS with Radeon Graphics

References

- ChatGPT: https://chatgpt.com/

Appendix

- GitHub: https://github.com/sophie8909/High-performance-programming