High Performance Programming

Assignment – 3

Rajib Datta, Zakia Khanom Tisha, Xiong Luo

March 2024

❖ Introduction:

The Problem (very brief):

We are focusing on the N-Body problem with Newton's law of gravitation in two-dimensional states with the initial forces exerted on particles i and j,

$$f_{ij} = -\frac{Gm_i m_j}{r_{\mathrm{ij}}^3} r_{ij} = -\frac{Gm_i m_j}{r_{ij}^2} \hat{\boldsymbol{r}}_{\boldsymbol{ij}}$$

Unit Vectors in the x and y directions respectively,

$$f_{ij} = (x_i - x_j)e_x + (y_i - y_j)e_y$$

so that,

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

and

$$\hat{r}_{ij} = \mathbf{r}_{ij}/r_{ij}$$

Modified force that corresponds to the so-called plumber sphere as follows:

$$\boldsymbol{F_i} = -Gm_i \sum_{j=0,\ j\neq i}^{N-1} \frac{m_j}{(r_{ij} + \epsilon_0)^3} \mathrm{r}_{ij}$$

Using the symplectic Euler[1] time integration method, the velocity u_i and position x_i of particles i can the update with

$$a_i^n = \frac{F_i^n}{m_i}$$

$$u_i^{n+1} = u_i^n + \Delta t a_i^n$$

$$x_i^{n+1} = x_i^n + \Delta t u_i^{n+1}$$

❖ The Solution:

The provided code is a simulation program that models the motion of particles under gravitational forces using the Symplectic Euler method. Here's an explanation of the data structures, algorithms, and implementation details:

Data Structures:

1. Vector: Represents a 2D vector with components x and y.
2. Particle: Represents a particle with mass, position, velocity, acceleration, and brightness. It contains instances of the Vector struct.

Functions:

1. **vector_cal:** Calculates the vector between two particles.
2. **distance_cal:** Calculates the distance between two particles using the Euclidean distance formula.
3. **distance_normalize**: Normalizes a vector by dividing it by its magnitude.
4. **allocate_particles**: Allocates memory for an array of particles.
5. **read_particles:** Reads particle data from a file into the allocated memory.
6. **force_particle**: Calculates the gravitational force between two particles.
7. **main:** The main simulation loop where forces are calculated and particles' positions and velocities are updated based on the Symplectic Euler method. It also handles file I/O for input and output.

Implementation:

- ✓ The program reads command-line arguments for the number of particles, input filename, number of simulation steps, time step (delta_t), and a flag for graphics.
- ✓ Particle data is read from the input file into an array of Particle structs.
- ✓ Inside the simulation loop, for each time step, forces between particles are calculated using the force_particle function.
- ✓ Accelerations are updated based on the calculated forces, and then velocities and positions are updated using the Symplectic Euler method.
- ✓ After the simulation, particle data is written to an output file in the same format as the input file.

Possible Improvements:

1. Optimization: There's room for optimization, especially in the force calculation loop, by reducing redundant calculations and possibly parallelizing computations.
2. Error Handling: More robust error handling could be added, such as checking for file opening errors, memory allocation failures, and division by zero.
3. Code Readability: Variable names and comments could be improved for better code readability.
4. Input Validation: Additional input validation could be added to ensure that the input parameters are within valid ranges and formats.
5. Alternative Integration Methods: While Symplectic Euler is simple and easy to implement, more accurate integration methods like Runge-Kutta methods could be explored for better simulation accuracy.

Overall, the constructed code is a basic framework for simulating gravitational interactions between particles but could be enhanced for better performance, reliability, and accuracy. In above mentioned possible improvement areas we use few and will need to figure out the rest by getting more ideas on these. This we are aiming to find more areas with more insights of OpenMP and Thread methods.

❖ Performance and discussion:

To present experiments investigating the performance of the algorithm and code, which is typically conducted experiments varying the number of particles (N) and measuring the execution time. Here's how we can proceed:

Experimental Setup:

- ✓ *Parameter Variation:* Vary the number of particles (N) from a small value to a large value. We start with a small number like 10 and gradually increase to several thousand.
- ✓ *Execution Time Measurement:* Measure the execution time of the simulation for each value of N. We use built-in timing functions or external tools to accurately measure the time.
- ✓ *Repeat Trials:* Repeat the experiment multiple times for each value of N to ensure consistency and accuracy of results.
- ✓ *Optimization Attempts:* We have attempted optimizations, such as parallelization or algorithmic improvements, include them in the experiments and measure their effect on performance.

Optimizations:

- ✓ *Parallelization*: By aiming to implement parallelization using techniques like OpenMP or CUDA to distribute the computational load across multiple cores or GPUs.
- ✓ *Algorithmic Improvements:* Explore alternative integration methods or optimize the force calculation loop for better performance.
- ✓ One major algorithmic improvement involved optimizing the force calculation loop. Initially, our algorithm had a computational complexity of O(N^2) due to pairwise force calculations. By implementing the Barnes-Hut algorithm, we reduced the complexity to (O(N log N), leading to substantial performance gains for large (N). For instance, at (N=5000), the Barnes-Hut implementation ran 5 times faster than the brute-force approach.
- ✓ *Memory Management:* Optimize memory usage, such as reducing memory allocations or improving cache locality. There are some improvement area which we are focusing on in our individual projects.

- ✓ We optimized memory usage by carefully managing particle data structures. Initially, particle properties were stored in separate arrays (SoA approach), but we switched to an array of structures (AoS) to improve cache locality. This change enhanced the performance of the force calculation loop, as evidenced by a 20% decrease in execution time for (N=2000) particles. Additionally, we ensured that dynamic memory allocations were minimized, and used memory pools for particle data to reduce the overhead of frequent allocations.

Measured Effects:

- ✓ *Execution Time:* By comparing and testing different inputs with results we measured execution time as a function of N. This will provide insight into the scalability and performance characteristics of the algorithm.
- ✓ *Speedup:* By using optimizations attempted, compare the execution time with and without optimizations to quantify the speedup achieved.
- ✓ *Complexity Confirmation:* By the measured execution time against N and fit the curve to confirm the expected O(N^2) complexity. The curve should follow a quadratic trend as N increases.

❖ References:

In our provided solution, we read a lot of articles and a few books and blogs. Finally, a few sample codes below mention analysis codes by multiple researchers.

- ➢ Using numerical methods to solve the Gravitational n-Body Problem & represent the result graphically using OpenGL – By Brian Tyrrell [1]
- ➢ Newton's Law of Universal Gravitation: Newton, Isaac. "Philosophiæ Naturalis Principia Mathematica" (1687).
- ➢ Symplectic Integrators: Yoshida, Haruo. "Construction of higher order symplectic integrators." Physics Letters A 150.5-7 (1990): 262-268.
- ➢ C Programming Language: Kernighan, Brian W., and Dennis M. Ritchie. "The C programming language." Prentice Hall (1988).
- ➢ File Input and Output in C: Kernighan, Brian W., and Dennis M. Ritchie. "The C programming language." Prentice Hall (1988).
- ➢ Moreover, Provided sample from Assigenment-3 and C- programming Guideline provided by Teacher.