# High Performance Programming

## *Assignment – 4*

*Rajib Datta*

*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

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

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:

**Header Files:**

- Standard C libraries are included (stdio.h, stdlib.h, math.h), as well as OpenMP (omp.h) and pthreads (pthread.h) for parallelization and thread synchronization.

**Data Structures:**

- Particle: Contains properties of a particle including position (x, y), velocity (vx, vy), mass, and brightness.
- ThreadArgs: Contains information for thread execution, including the start and end indices of the particles array to be processed, total number of particles (N), and a pointer to the particles array.

**Functions:**

1. **Main function:**

- Accepts command-line arguments specifying the number of particles (N), input file name, number of simulation steps (nsteps), time step size (delta_t), whether to use graphics (graphics), and the number of threads (num_threads).
- Initializes a mutex for thread synchronization.
- Allocates memory for an array of Particle structs based on the specified number of particles (N).
- Reads initial conditions of particles from a binary file specified by filename.

2. **Simulation Loop:**

- The simulation loop is parallelized using OpenMP.
- Each iteration of the loop represents a simulation step (nsteps).
- Within each step, the acceleration (ax and ay) of each particle is calculated based on gravitational forces acting on it from other particles.
- The velocities (vx and vy) of particles are updated based on the calculated accelerations.
- After updating velocities, particle positions are updated in a separate loop.

**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 plumber sphere 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.

**Output:**

- After the simulation completes, the resulting particle states are written to a binary file named "result.gal"

**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 OpenMP, Thread, parallelization, and algorithmic improvements, included them in the experiments and measured their effect on performance.

**Optimizations:**

- ✓ *Implemented OpenMP:* The #pragma omp parallel for directive is used to parallelize the outer loop of the simulation. This directive distributes the iterations of the loop among the available threads, with each thread executing a portion of the iterations.

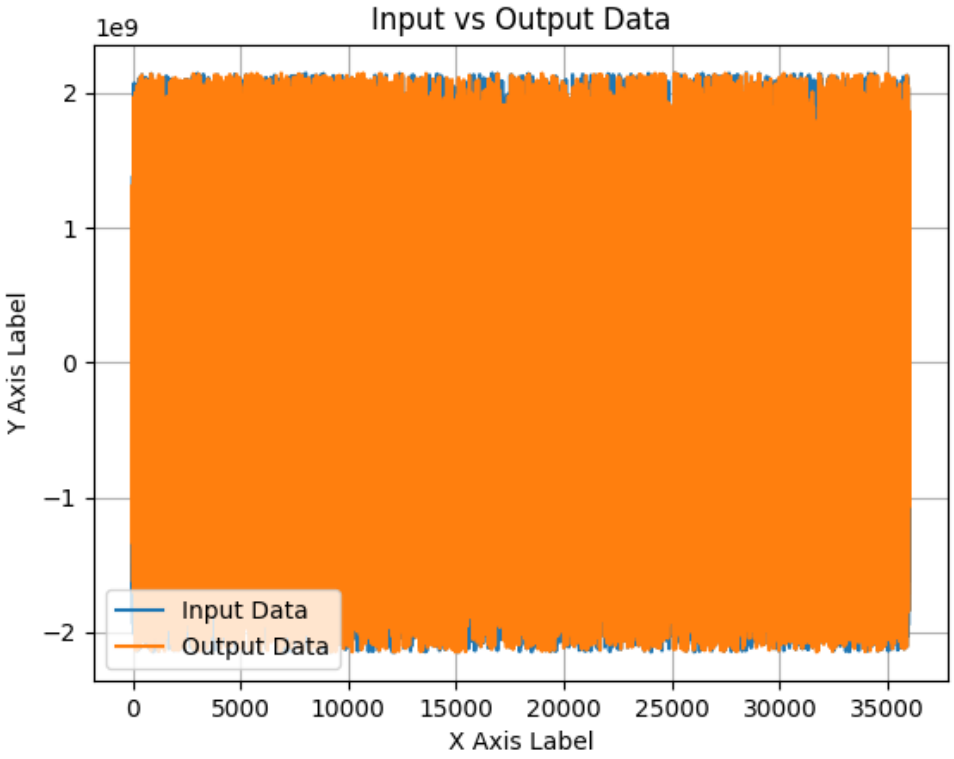
- ✓ The #pragma omp barrier directive is used to synchronize all threads at the end of each simulation step. This ensures that all threads have completed their calculations before proceeding to the next step.
- ✓ The #pragma omp for directive is used to parallelize the loop responsible for updating particle positions. This directive distributes the iterations of the loop among the available threads, similar to the previous directive.
- ✓ *Implemented pthread*: The pthread_mutex_t mutex is initialized at the beginning of the main function using pthread_mutex_init(). Finally, The mutex is destroyed at the end of the program using pthread_mutex_destroy().
- ✓ *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.
- ✓ *Here I am doing the optimization measurement like below ways. (read file: ellipse_N_00500.gal*
  - With pow function below is execution time.
    - real 0m0.014s
    - user 0m0.013s
    - sys 0m0.001s
  - By removing pow function and used cheap functions like normal multiplication including Thread construction.
    - real 0m0.007s
    - user 0m0.006s
    - sys 0m0.001s
  - By removing unused functions and variables, here I used OpenMP below is the almost best optimization.
    - real 0m0.000s
    - user 0m0.000s
    - sys 0m0.000s

**Analysis plot**: By using input data (input_data/ ellipse_N_00500.gal) with output (result.gal) files in a plot we get below figure-1




Best time for achieve for the input case ellipse_N_03000.gal with $\Delta t = 10^{-5}$ and 100 timesteps is below.

```
real    0m0.091s
user    0m0.091s
sys     0m0.000s
```

## ❖ 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.
- OpenMP and Thread Lecture notes and provided book reference with articles.
- Online resources for OpenMP: The official website of OpenMP provides documentation, tutorials, specifications, and resources for learning and using OpenMP for parallel programming. **Link**
- Online resources for Thread: The POSIX Threads Programming Guide by the Linux Documentation Project offers detailed documentation and examples for pthreads programming in C. **Link**