Parallelization of N-body Simulation

Kai-Hsun Lu
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
jameslu.ii14@nycu.edu.tw

Wei-Lin Wen
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
weilin.cs14@nycu.edu.tw

Wei-Ting Yu
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
ywt.cs14@nycu.edu.tw

1 Introduction / Motivation

The motivation of this project is to accelerate a physical simulation using various parallel programming techniques and to theoretically validate the expected performance improvements with practical results, so we selected the three-body simulation as a starting point and extended it to the more general N-body simulation.

The N-body problem models the evolution of a system of particles under mutual forces, however, the computational complexity of direct pairwise force evaluation is O(N^2), making simulations with large numbers of particles computationally expensive.

To address this challenge, we aim to significantly accelerate N-body simulations and analyze the resulting performance gains with modern compute devices and various parallel programming techniques.

2 Statement of the Problem

Application chosen: N-body simulation (gravitational interactions).

Reason for choice:

(1) The computation workload in N-body simulation is highly parallelizable by nature, allowing the use of different optimization approaches.
(2) The computational complexity of O(N^2) makes it a meaningful challenge, providing a strong motivation to explore parallelism, vectorization, and efficient workload distribution.
(3) The original implementation uses SDL for rendering, which introduces an opportunity to further investigate rendering parallelization and improve visualization performance alongside the computation.

3 Proposed Approaches

We will implement and compare multiple parallelization strategies for N-body simulation and introduce Tracy instrumentation methods to measure performance characteristics, identify bottlenecks, and evaluate the effectiveness of different parallel computing approaches:

(1) Accelerations:
- serial: orignal implementation.
- Pthread pair: Directly parallelize pairwise force computations.
- Pthread balanced: Directly parallelize but dont compute by pairwise.
- Pthread interleaved: parallelize pairwise force computations but with interleaved index
- Mutex versions: extend all above pthread implementations to persist threads and synchronize shared data using mutexes.
- SIMD versions: apply SIMD intrinsics to all above implementations to exploit data-level parallelism.
- Task based parallelizm
- GPU acceleration: offload force computations to GPU using CUDA/OpenCL for massive parallelism.

(2) step leapfrog integration:
- Identical parallelization strategies as above

(3) rendering:
- SDL (Simple DirectMedia Layer): The original implementation uses SDL to visualize particle positions and trajectories.
- While SDL rendering functions generally must be executed on the main thread and are not inherently parallelized, we can still parallelize certain pre-rendering tasks, such as offscreen buffer generation, where frames are computed in a memory buffer in parallel before being displayed, if time permits.

3.1 Block Diagram / System Architecture

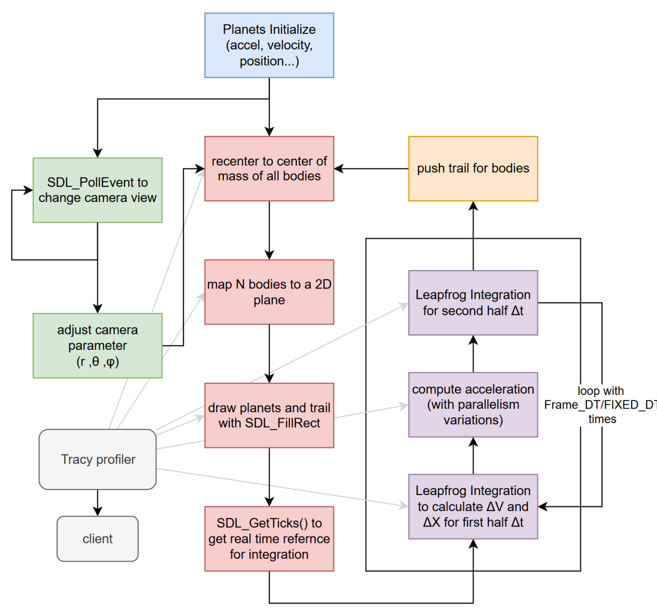

Figure 1: System Architecture of the N-Body Simulation

3.2 Component Functions

(1) **Initialization Components:**
- **Planet Initialization:** Sets up N bodies with random positions and velocities around a center point
- **Trail Initialization:** Creates empty trail buffers for each body to track motion history
- **SDL Setup:** Initializes graphics window and surface

(2) **Physics Integration Components:**
- **`step_leapfrog()`:** Implements leapfrog integration algorithm with kick-drift-kick pattern: updates velocities by half-step, updates positions, recalculates accelerations, then updates velocities by another half-step
- **`accelerations()`:** Computes gravitational forces between all bodies

(3) **Coordinate Management Components:**
- **`recenter()`:** Calculates center of mass and shifts all bodies so the center of mass is at screen center, preventing the system from drifting off-screen

(4) **Trail Management Components:**
- **`trail_push()`:** Adds new position to circular trail buffer only if body moved at least `MIN_DIST` from last recorded position
- **`trail_draw()`:** Renders motion trails with depth-based brightness

(5) **Rendering Components:**
- **`fill_circle()`:** Draws planets as filled circles with perspective scaling and brightness based on z-depth
- **Screen Update:** Clears screen, draws all trails and planets, updates display

(6) **Event Handling Components:**
- **`SDL_PollEvent()`:** Checks for quit events (window close) and handles camera control inputs for adjusting spherical coordinates (r, θ, φ)

(7) **Time Management Components:**
- **Frame Timer:** Uses `SDL_GetTicks()` to measure real elapsed time
- **Fixed Timestep Accumulator:** Ensures physics runs at consistent `FIXED_DT` intervals regardless of frame rate

3.3 Component Interactions

(1) **Physics Update Loop:** Calculate `frame_dt` and add to accumulator and while accumulator ≥ dt, call `step_leapfrog(bodies, FIXED_DT)` which updates velocities (half-step), positions, accelerations, and velocities (half-step), then decrease accumulator
(2) **Coordinate Management:** `recenter(bodies)` calculates center of mass and shifts all bodies to screen center
(3) **Depth Effect:** Both `fill_circle()` and `trail_draw()` use z-coordinates to create perspective through size scaling and brightness adjustment
(4) **Trail Recording:** For each body, `trail_push()` adds current position to trail buffer
(5) **Rendering:** Clear screen, then for each body call `trail_draw()` and `fill_circle()`, finally `SDL_UpdateWindowSurface()`
(6) **Tracy Integration:** `ZoneScopedN()` calls throughout enable performance profiling without affecting logic

3.4 Mathematical Formulations

Leapfrog Integration (kick-drift-kick pattern):

$$\mathbf{v}_{i+1/2} = \mathbf{v}_i + \frac{1}{2}\mathbf{a}_i \Delta t \tag{1}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1/2}\Delta t \tag{2}$$

$$\mathbf{a}_{i+1} = f(\mathbf{x}_{i+1}) \tag{3}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_{i+1/2} + \frac{1}{2}\mathbf{a}_{i+1}\Delta t \tag{4}$$

Center of Mass Calculation:

$$\mathbf{r}_{COM} = \frac{\sum_{i=1}^{N} m_i \mathbf{r}_i}{\sum_{i=1}^{N} m_i} \tag{5}$$

$$\mathbf{r}_i' = \mathbf{r}_i + (\mathbf{r}_{screen} - \mathbf{r}_{COM}) \tag{6}$$

Spherical Camera System: The camera position is defined in spherical coordinates (r, θ, φ) where r is the radial distance, θ is the azimuthal angle, and φ is the polar angle.

Conversion to Cartesian coordinates:

$$x_{cam} = r\cos\theta\cos\varphi \tag{7}$$

$$y_{cam} = r\sin\theta\cos\varphi \tag{8}$$

$$z_{cam} = r\sin\varphi \tag{9}$$

3D to 2D Projection: Map body positions to camera view using perspective projection:

$$\mathbf{r}_{rel} = \mathbf{r}_i - \mathbf{r}_{cam} \tag{10}$$

$$x_{screen} = \frac{f \cdot x_{rel}}{z_{rel} + d} + \frac{W}{2} \tag{11}$$

$$y_{screen} = \frac{f \cdot y_{rel}}{z_{rel} + d} + \frac{H}{2} \tag{12}$$

where f is the focal length, d is the view distance, and (W, H) are screen dimensions.

4 Language Selection

- **C++ with pthread** for shared-memory multiprocessing, providing efficiency and fine-grained control over memory and performance with support for instrumentation tools like Tracy.
- **CUDA** for GPU acceleration, well-suited for massively parallel workloads like N-body simulations by directly mapping particles to GPU threads for concurrent force calculations.

5 Related Work

- Francesco, L (2025). C-projects [Source code]. GitHub. https://github.com/mrparsing/C-Projects
- Bartosz, Taudul (2025). tracy [Source code]. GitHub. https://github.com/wolfpld/tracy
- Rein van den Boomgaard (2017). Image Processing and Computer Vision, The Pinhole Camera

6 Expected Results

- Parallel results will accurately match the original implementation.
- Parallel implementations are expected to outperform the serial version.

- Parallel methods scale well and show speedup for different N (1K, 10K, 100K bodies).
- Performance assumptions match with Tracy measurements theoretically and mathematically.

7 Timetable

Week	Task
3	1. Literature review and project planning 2. Implement baseline sequential N-body simulation
4	1. Integrate Tracy profiling framework 2. Implement pthread parallelization
5	Optimize pthread implementation and profiling
6	Implement CUDA GPU acceleration
7	Optimize CUDA implementation and profiling
8	Implement spherical camera system
9	Performance comparison and bottleneck analysis
10	Refinement and optimization of all implementations
11	1. Run experiments, collect performance data 2. Analyze results, prepare visualization
12	Write final report and presentation

8 References

(1) Hockney, R. W., & Eastwood, J. W. (1988). *Computer Simulation Using Particles*. CRC Press. (Leapfrog integration and N-body algorithms)

(2) NVIDIA Corporation. (2023). *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/

(3) Tracy Profiler Documentation. (2024). https://github.com/wolfpld/tracy/releases/latest/download/tracy.pdf

(4) Butcher, P., & Devlin, J. (2011). *Pthreads Programming: A POSIX Standard for Better Multiprocessing*. O'Reilly Media.

(5) SDL (Simple DirectMedia Layer) Documentation. (2024). Retrieved from https://wiki.libsdl.org/

(6) Bermudez-Cameo, Jesus, Lopez-Nicolas, Gonzalo, Guerrero, Josechu. (2012). A Unified Framework for Line Extraction in Dioptric and Catadioptric Cameras.

(7) Quinn, T., Katz, N., Stadel, J., Lake, G. (1997). Time stepping N-body simulations. https://arxiv.org/abs/astro-ph/9710043