CSEE4840 Project Proposal: Large-Scale N-Body Accelerator

Lucy He lh3365 | Xiyuan Peng xp2236 | Jingzeng Xie jx2668 | Pengpeng Wang pw2660 | Charlotte Chen hc3558

2026 Spring

1 Introduction

The goal of this project is to design and implement a hardware accelerator for gravitational N-body simulation on FPGA. N-body simulation is a classic computational problem in which each body interacts with every other body through pairwise forces, leading to $O(N^2)$ computational complexity. It can be used in various applications, including astrophysics, climate models, plasma and particle systems, and interactive physics engines for games and animation. Because the pairwise force calculations are highly regular and parallelizable, the problem is well-suited for a custom hardware pipeline.

Our proposed system focuses on accelerating the force computation and time-update loop for a 2D gravity simulation. The hardware accelerator will consist of four parallel two-body compute cores arranged in a systolic-style pipelined data path. Each core will compute pairwise gravitational contributions using a fast inverse square root unit, followed by calculation and accumulation. A leapfrog integration step will then update particle velocities and positions over time. On the software side, the host processor will manage initialization, batching of body data, and display/output coordination so that simulations with larger numbers of bodies can be supported without requiring the entire problem state to reside in on-chip FPGA memory.

For demonstration, we plan to render the simulation on a VGA display as an interactive 2D gravity simulator with moving particles for up to 2000 bodies at 30 FPS.

2 N-Body Computation and Numerical Method

The main computational task is to evaluate the gravitational acceleration on each body due to all other bodies. For body i, the acceleration is computed as

$$\mathbf{a}_i = \sum_{j \neq i} G m_j \frac{\mathbf{r}_{ij}}{(|\mathbf{r}_{ij}|^2 + \epsilon)^{3/2}}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, m_j is the mass of body j, and ϵ is a small softening term used to avoid numerical singularities when two bodies are very close together(1).

A direct all-pairs implementation is expensive in software, but maps naturally onto a parallel FPGA architecture. In our design, multiple pairwise interactions will be evaluated concurrently by four two-body cores. Each core will process one body pair at a time and output partial acceleration contributions. These contributions will be accumulated to obtain the total acceleration for a target body.

For time integration, we propose to use the leapfrog integrator to update particle positions and velocities because it provides a good balance between numerical stability and hardware cost for gravitational N-body simulation. In kick-drift-kick form, leapfrog updates are

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + \mathbf{a}(\mathbf{x}_n)\Delta t, \qquad \mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{v}_{n+\frac{1}{2}}\Delta t.$$

Compared to Euler integration, leapfrog is second-order accurate rather than first-order, and it is also time-reversible, which helps suppress long-term numerical dissipation. Compared to RK4, leapfrog requires far less arithmetic and no storage of multiple intermediate stages, making it much more suitable for a pipelined FPGA implementation. For large gravitational simulations, this gives leapfrog an attractive combination of efficiency and better long-term energy behavior than conventional methods such as Euler and RK4 (2).

3 Hardware Implementation

The hardware portion of the project will implement a parallel pipelined N-body accelerator. The core compute engine will contain four two-body processing cores. Each core will take as input the state of two bodies, including position and mass, and will compute one pairwise gravitational contribution. The main arithmetic stages are:

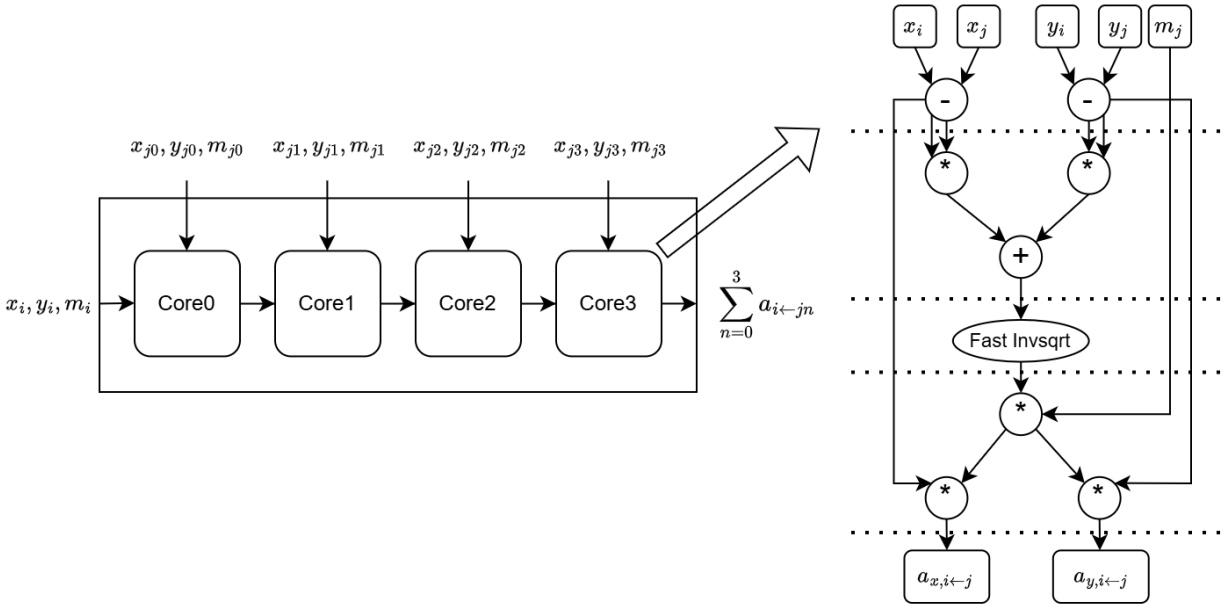

Figure 1: Hardware block diagram

1. displacement calculation: compute Δx and Δy
2. squared distance calculation: compute $r^2 = \Delta x^2 + \Delta y^2 + \epsilon$
3. fast inverse square root: approximate $1/\sqrt{r^2}$
4. force scaling: form $1/r^3$ and multiply by mass and displacement
5. accumulation: add the partial acceleration contribution into the running sum

The fast inverse square root block is a key part of the design, since gravitational force requires inverse distance cubed. Using a hardware-oriented inverse square root approximation substantially reduces computational cost compared to a fully general division or square root unit, while still providing enough accuracy for a real-time visual simulation(3).

At a higher level, the four two-body cores will be used in a streaming architecture. Body data will be fed through the pipeline in batches, allowing one target body to interact with several source bodies in parallel. Partial sums will be accumulated and passed forward until the full acceleration for the target body is complete, after which the leapfrog update will produce the next position and velocity state.

4 Hardware-Software Interface

A key challenge of the project is supporting a larger number of bodies than can be conveniently stored and updated entirely inside FPGA on-chip memory. To address this, we plan a hardware-software co-design approach in which the software manages the global simulation state and the FPGA acts as a streaming accelerator.

The software will store the full list of bodies and send body data to the FPGA in batches. For each batch, the FPGA will compute pairwise force contributions and return either updated body states or accumulated accelerations. The software can then recombine the returned results and schedule the next batch. This batching approach allows us to scale to larger N without requiring large BRAM allocations for the entire simulation history or full all-pairs matrix.

The software side will also handle initialization of simulation parameters, selection of predefined scenes, and movement of data between memory and the hardware accelerator. This division of labor keeps the FPGA focused on the arithmetic-intensive inner loop while allowing the software to provide flexibility in problem size, configuration, and visualization control.

5 Demo and User Interface

Our planned demo is a VGA-based real-time gravity simulator. Bodies will be displayed as moving particles on screen, with optional trails to visualize orbital paths. We expect to support several predefined scenarios, such as two-body orbit, multi-body orbital systems, and random particle clusters. We will also include simple runtime controls such as pause, reset, timestep adjustment, or switching between different initial configurations.

The main purpose of the demo is to clearly show that the hardware accelerator is computing physically meaningful motion in real time, and that the parallel compute pipeline provides enough throughput to support interactive visualization.

6 Major Tasks and Milestones

The main tasks for the project are:

1. Define the numerical format and architectural parameters, including coordinate range, timestep, softening factor, and whether the arithmetic is fixed-point or reduced-precision floating point.
2. Design and verify a single two-body core, including displacement, inverse square root, force computation, and accumulation.
3. Replicate and integrate four two-body cores into a parallel pipelined accelerator architecture.
4. Implement leapfrog integration and state update logic.
5. Develop the hardware-software interface for batch transfer of body data and collection of results.
6. Implement VGA visualization and prepare several demonstration scenes.
7. Evaluate performance, resource usage, and numerical behavior of the final system.

7 Potential Issues and Open Questions

First, numerical representation will strongly affect both accuracy and resource usage. Fixed-point arithmetic may be more efficient on FPGA, but care is needed to preserve sufficient dynamic range for positions, velocities, and inverse-distance terms. Reduced-precision floating point is another possibility if fixed-point scaling becomes too restrictive.

Second, the batching strategy between software and hardware must be designed carefully so that communication overhead does not dominate the acceleration benefit. Since the system will not store a very large number of bodies entirely on chip, the effectiveness of the project depends on achieving a practical streaming interface.

Third, the overall dataflow and accumulation schedule must be chosen so that four two-body cores are well utilized. A central architectural question is how best to map a large all-pairs interaction pattern onto a small number of reusable compute cores while maintaining pipeline efficiency.

Finally, VGA rendering and simulation throughput must be balanced. The demo does not require physically perfect scientific simulation, but it should clearly demonstrate stable and believable gravitational motion at interactive speed.

8 Expected Outcome

By the end of the project, we aim to demonstrate a working FPGA-based N-body gravity simulator with:

- four parallel two-body compute cores,
- pipelined inverse-square-force computation,
- leapfrog integration for time evolution,
- software-managed batching for larger simulations, and
- real-time VGA visualization of moving bodies.

The project will serve as a concrete demonstration of how a regular physics workload can be mapped efficiently to a custom FPGA accelerator through parallelism, pipelining, and hardware-software co-design.

References

[1] E. D. Sozzo, M. Rabozzi, L. D. Tucci, D. Sciuto, and M. D. Santambrogio, "A scalable fpga design for cloud n-body simulation," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Milan, Italy, 2018, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8445106

[2] S. Tremaine, "Numerical integration in celestial mechanics," 2014, lecture notes, Institute for Advanced Study. [Online]. Available: https://www.ias.edu/sites/default/files/sns/Numerical-integration%282%29.pdf

[3] F. Stokes, "Fast inverse square root," GitHub Blog, May 2024, [Online]. Available: https://github.com/francisrstokes/githublog/blob/main/2024%2F5%2F29%2Ffast-inverse-sqrt.md#newtons-method.