N-Body Simulation on an Altera DE2-115 FPGA

Enrique Gomez (exg), Maxwell Guo (mwguo)
https://mwguo15.github.io/n-body-on-fpga/

Summary

We propose to build a n-body simulator using systolic arrays entirely on an FPGA. Our engine will use a connected line of processing elements to feed streaming bodies across the entire pipeline, calculating inter-body forces along the way. All data will be stored in the FPGA's BRAM and sent back to a PC via UART to verify correctness. For comparison, we will also be implementing an optimized n-body simulator on a GPU with CUDA. The goal is to achieve comparable or even greater performance with the FPGA by leveraging pipelining, DSP blocks, and memory optimizations.

Background

In n-body simulation, a body's velocity and position are updated every time step to simulate how large bodies (planets, stars) interact in our universe. However, these velocity and position updates are based on the forces exerted by all other bodies in the system. Sequentially calculating all the forces on a single body requires iterating over all the other N - 1 bodies. This results in $O(n^2)$ compute time for the naive approach. While this is an extremely parallel algorithm, the amount of resources required to execute a fully correct, parallel approach over millions of bodies is impractical. This is why there are algorithms that cut down the amount of necessary computation by approximating results. For instance, the Barnes-Hut approach groups bodies (approximation to reduce computation) and stores these groups in trees, resulting in O(nlogn) time complexity. Due to the hardware and programming limitations of FPGAs, implementing an algorithm like Barnes-Hut would be extremely complex.

The Challenge

As stated previously, the n-body simulation problem is inherently computationally intensive because its naive formulation scales quadratically with the number of bodies. Mapping such an algorithm onto an FPGA presents several key challenges:

Computational Complexity and Parallelism:
The primary challenge is efficiently handling the massive number of arithmetic operations required to compute gravitational forces. Even with moderate n (e.g., several hundred to a few thousand bodies), the number of pairwise computations quickly becomes overwhelming. Although FPGAs excel at parallel processing, fully unrolling the $O(n^2)$ workload is impractical

due to resource limitations. Our baseline “naive” design must therefore rely on time-multiplexed and pipelined arithmetic units. Exploring advanced parallel architectures, such as systolic arrays, introduces another layer of complexity: we must carefully design processing elements (PEs) that can communicate locally in a pipelined fashion while minimizing global interconnect delays and resource overhead.

Fixed-Point Arithmetic and Precision Trade-Offs:
The DE2-115 Cyclone IV supports fixed-point arithmetic, which forces us to carefully balance numerical precision and hardware efficiency. Multiplications are efficiently mapped to the 266 embedded 18×18 multipliers; however, operations like division (or reciprocal approximation) and accumulation of many small contributions can introduce rounding errors. Ensuring numerical stability over many simulation steps—and verifying the results against a high-precision CPU or GPU reference—is a non-trivial challenge that requires both clever hardware design and rigorous testing.

Memory Access and Data Movement:
Although the FPGA’s 3,888 Kb of embedded memory (BRAM) is sufficient to store the state for thousands of bodies, the challenge lies in managing memory bandwidth and data movement. In a naive implementation, every body’s state might be read and updated repeatedly, leading to potential bottlenecks. A systolic array design mitigates this by localizing communication between PEs, yet it imposes its own challenges in synchronizing the data flow and ensuring that each PE receives the correct data at the right time without excessive buffering overhead.

Scalability and Optimization:
Our project starts with a naive approach to the n-body problem, but we plan to incorporate systolic arrays or similar optimizations to scale up the number of bodies simulated. Designing such an array involves complex control logic (FSMs) and careful pipeline balancing to achieve high throughput without exceeding the available FPGA resources (logic elements, DSP blocks, and BRAM). If time permits, we also aim to explore approximate methods—such as grid-based interactions—to further reduce the quadratic computational cost. Understanding the trade-offs between accuracy, resource usage, and computational throughput is a central challenge of this project.

Verification and Debugging:
Verifying the correctness of the simulation, especially when scaling to thousands of bodies, is a substantial challenge. The iterative nature of the simulation combined with fixed-point arithmetic can lead to subtle errors that accumulate over time. We plan to implement robust verification strategies—including memory-mapped readouts, automated regression testing, and cross-validation with GPU/CPU implementations, to ensure that our optimized design not only

achieves similar/better computation time but also produces accurate results.

Overall, our project will push us to deeply understand and address the challenges of parallelizing a computationally heavy workload on an FPGA, optimizing both dataflow and arithmetic operations, while also providing a great opportunity to explore systolic array architectures as a means to overcome the inherent $O(N^2)$ complexity of the n-body simulation.

Resources

Our development and testing will primarily occur on the ECE lab machines in HH1305, which are equipped with VCS (Verilog Compiler Simulator) and Intel Quartus (HDL Synthesizer). These tools will allow us to simulate our designs for testing and then synthesize our design onto the FPGA. If time permits for further optimizations, we will also use the Synopsys Design Compiler for critical path, power, and area analysis. As for the FPGA, we will borrow the Altera DE2-115 Cyclone IV from a previous professor. Development and testing for the CPU + GPU implementation will primarily occur on our personal machines, which are equipped with NVIDIA GPUs (an RTX 3060 and an RTX 3070). We will also leverage the GHC lab machines with RTX 2080 GPUs for further performance evaluation.

Goals and Deliverables

By the end of the project, we plan to deliver a working FPGA n-body simulation along with a thorough performance analysis. Below we outline our expected outcomes:

- **Core Deliverables (Plan to Achieve):** Correct, working FPGA implementation of an optimized n-body simulation. This would involve designing the full systolic array architecture that pipelines all n^2 pairwise computations. This implementation would require N memory accesses per pass over ceil(N/K) passes, where K is the number of processing elements (PEs) in the systolic array. Each PE will compute force contributions for one body. This minimizes the amount of memory accesses and utilizes the DSP blocks better. We hope to compare performance metrics (resource utilization, energy efficiency, speed) with the naive implementation. We also hope to compare speed and precision (fixed-point vs floating-point) with the GPU implementation. All of these comparisons will be represented with benchmarks, graphs, and a trade-off report.
- **Stretch Goals (Hope to Achieve):** In addition to the core deliverables, we would like to attempt scaling the design to handle 1,024–4,096 bodies by further optimizing the systolic array, using deeper parts of the memory hierarchy (SRAM, SDRAM), or by compressing the data. We would also like to experiment more with the number of PEs, clock speed, and other tunable parameters. Lastly, we would like to compare the performance to power (perf/watt) ratios for the FPGA and GPU implementations.

- **Minimum Viable Product:** Correct, working FPGA implementation of a naive n-body simulation. This would involve designing a datapath and FSM-based accelerator that computes n^2 pairwise computations through simple array iteration. This implementation would require the maximum amount of memory accesses and not utilize all 266 DSP blocks. Correctness would be fully verified via the GPU reference.

Platform Choice

We have chosen to program our FPGA with SystemVerilog and use C++ with CUDA for the CPU + GPU implementation. This is because we are most familiar with these languages and they are standard for their respective applications.

Schedule

Week 1 (April 7 – April 13):
In the first week, we will create a minimal, "naive" N-body simulator using SystemVerilog on the FPGA. This initial version will compute all pairwise forces by iterating over each body in turn, storing positions and velocities in the BRAM for quick access. We will also set up UART communication between the FPGA and a PC to facilitate data transfer for testing and verification. After synthesizing this naive design in Quartus, we will review resource utilization (logic elements, DSP blocks) and identify any immediate issues, such as the need for more efficient fixed-point arithmetic or changes in our memory access pattern. Throughout this process, we will cross-verify intermediate results against a smaller GPU or CPU reference to ensure correctness before moving on.

Week 2 (April 14 – April 20):
Next, our focus will shift to designing and partially integrating the systolic array architecture. We will create a pipeline of processing elements (PEs) that pass bodies along in a chain, each PE computing the incremental force contribution. The key tasks will include setting up the local buffering required to support the dataflow, ensuring that bodies arrive in the correct order, and addressing the challenges of fixed-point arithmetic for stable accumulations of forces. After establishing the pipeline, we will test our systolic array on small numbers of bodies (e.g., 16–64) by retrieving intermediate values over UART and comparing the outputs to a known CPU/GPU reference. We will also begin early optimization for pipeline timing, making sure we can meet timing constraints at our target clock frequency.

Week 3 (April 21 – April 28):
In the final week, we will integrate all remaining pieces and verify correctness at scale, pushing the design to handle larger numbers of bodies (e.g., 256–1,024, as resources allow). Here, we will finalize the numeric format and optimize for performance, monitoring resource utilization and possibly clock frequency scaling. We will conduct thorough benchmarking to compare our

FPGA design—both the naive and systolic-array versions—against the GPU implementation, focusing on throughput (updates per second), resource usage, and any measurable energy impact. Once we collect all these metrics, we will compile our results into graphs and analyses, which will serve as the basis for the final deliverables, including a detailed report on our design decisions, trade-offs, and performance outcomes.