FPGA-Accelerated N-Body Simulation

Team A3: Abhinav Gupta, Rene Ravanan, and Yuhe Zheng
18-500 Capstone Design, Fall 2023
Electrical and Computer Engineering Department
Carnegie Mellon University

Product Pitch

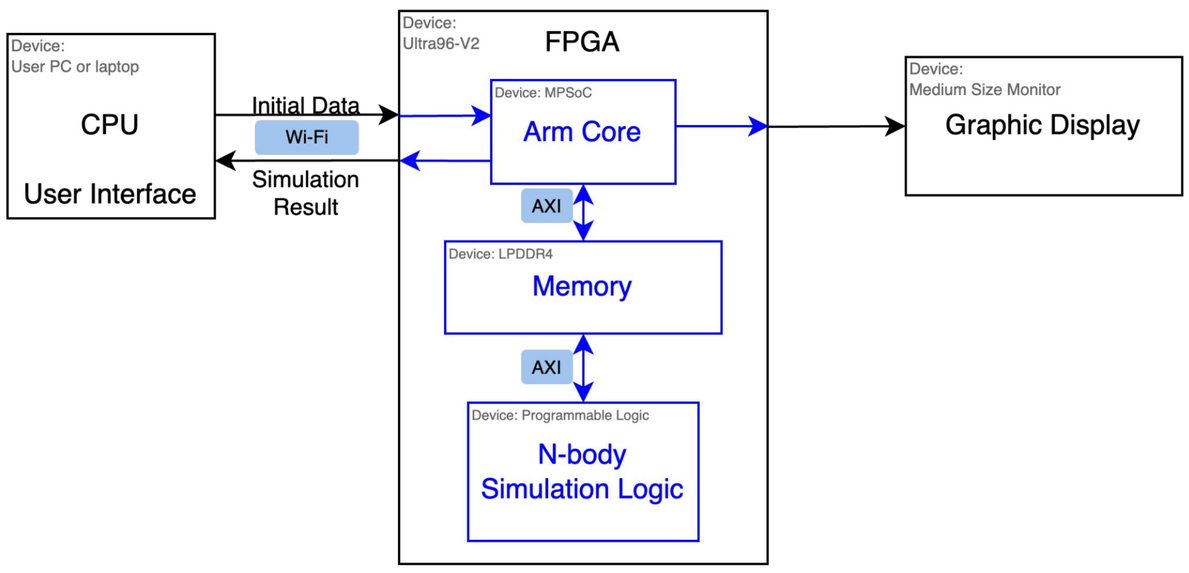
N-Body simulations are a computationally intensive problem suited for FPGA acceleration due to high workloads and high amounts of irregularity in computation. CPU implementations are not fast enough and GPU ones are not power efficient.

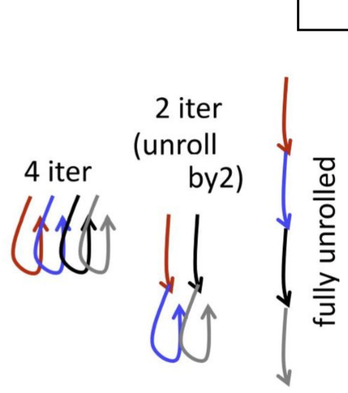
Our goal is to apply FPGA acceleration to the N-Body simulation. We want to provide at least a **10x speedup** compared to a CPU implementation with a **90-95%** accuracy.
We developed an FPGA based simulation system using Vitis HLS that accepts initial particle data as an input file on which it runs our simulations for a specified number of iterations. Results of these simulations are then stored in an output file and can also be viewed on a graphical display.

This product is suitable for physicists that are on a budget with time constraints and are trying to run astronomical/ molecular simulations efficiently.

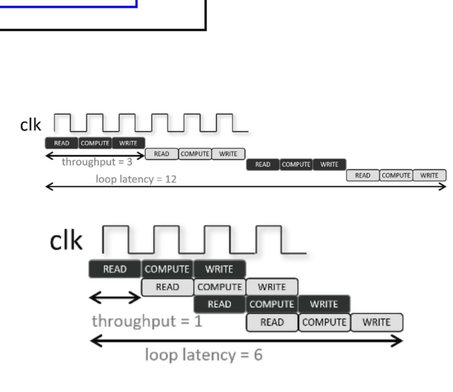
System Architecture

Our system operates primarily on the FPGA with our CPU interfacing with the board's ARM core to send and extract particle data. Once the data initialised, the FPGA transfers this data on to its memory and configurable logic via the highly efficient AXI Burst protocol (commonly used on many modern SoCs). Output data is also sent to our graphic display.

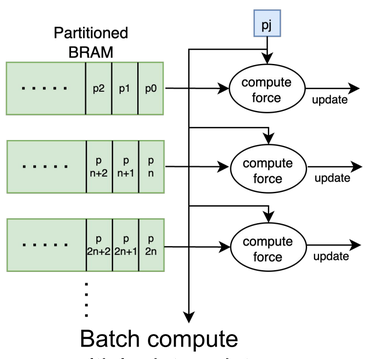




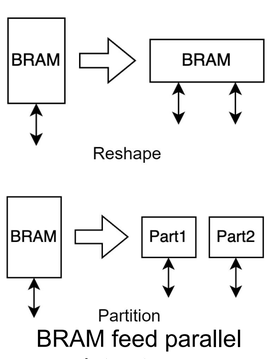
Unrolling



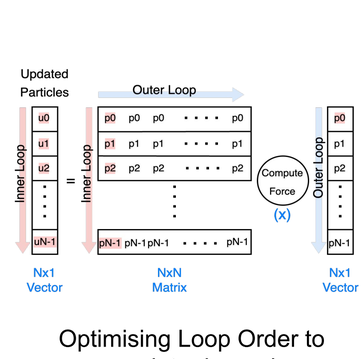
Pipelining



Batch compute multiple data points



BRAM feed parallel data stream



Optimising Loop Order to remove data dependency

Figure 1: Optimizations Summary

Conclusions & Additional Information

Overall we were able to meet our goals by a wide margin, which we are very proud of. We focused mainly on the technical aspects of the project, and put a lot of effort into getting a good speedup. Some approaches we tried that did not work include using fixed point types and DRAM port widening.

Our project combined the disciplines of software systems for the actual implementation of the N-Body simulation algorithm, and hardware systems for the FPGA optimizations. We were able to separately work on optimizing hardware computation model, porting the algorithm to the FPGA, and developing our graphical display. We work together on communication between different portions of the algorithm, integrating our systems/optimisations, and planning general design strategies.

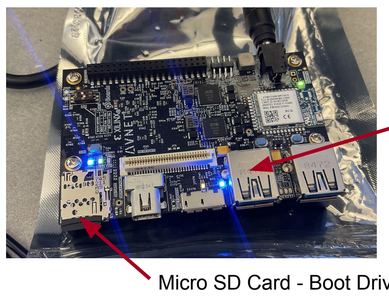
Further extensions to this project would be to try different hardware platforms in addition to Vitis, implementing our system directly in RTL instead of HLS to exploit more hardware, and to have a more polished user interface.

Scan to get read project details on our website.

https://course.ece.cmu.edu/~ece500/projects/f23-teama3/

Electrical & Computer ENGINEERING
Carnegie Mellon

System Description



Our system supports simulation of up to 10000 particles. The number of particles matches our user needs for astronomical and molecular simulations. This number of particles also fit the hardware resources available on our Ultra96v2 FPGA.

In our system, users prepare inputs off-board and are able to upload them via SCP. Then, the accelerated N-Body simulation kernel is able to run on these inputs and produce an output file that can be viewed/post-processed by the user offline. Simulation result of each iteration is also sent to a graphical display for visualization. Users have an option to either enable this visualisation to be synchronous (reducing our speedup) or asynchronous.

System Evaluation (Accuracy)

We used a CPU implementation which uses double precision floating point (this was in order to make sure that we are comparing ourselves to a model which is more accurate than us as we are using single precision floating point). Graph 1 describes the relationship between the number of iterations and the accuracy. As expected the accuracy of our model decreases and the number of iteration increaser as the small error which our model has propagates and increases overtime. We still maintain a good accuracy even when running our model for 1000 iterations.

Note - Even the CPU implementation is not fully accurate as it is impossible to fully contain position data even in 64-bit float, but we thought it was good enough to use as a reference value

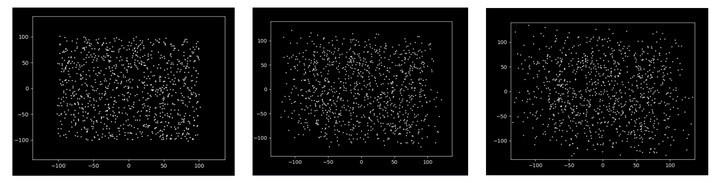
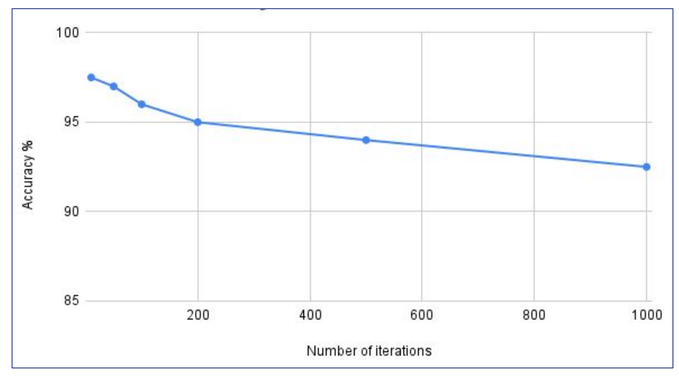


Figure 2: Graphics Results

$$\frac{\sum_{j=1}^{\text{Num Iter}} \sum_{i=1}^{10000} \left[\frac{\left|\frac{x_{ref}-x_{test}}{x_{ref}}\right| + \left|\frac{y_{ref}-y_{test}}{y_{ref}}\right|}{2} \right] \times 100}{\text{Num Iter} \times 10000}$$

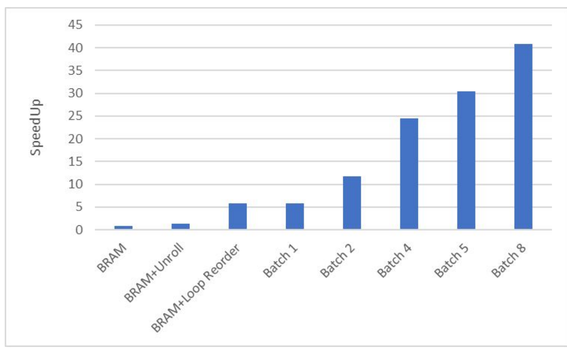
Figure 3: Formula used to calculate accuracy



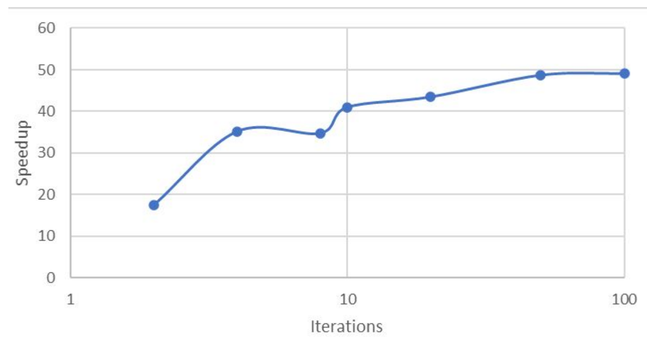
Graph 1: Accuracy vs Number of iterations

System Evaluation (Performance)

Observing the graphs below, as we apply the optimizations described in the System Architecture section, we see a decrease in the runtime of our kernel (speedup = CPU runtime / FPGA runtime). With just BRAM optimizations and loop refactoring alone we achieve a ~5x speedup, but with batching coupled with unrolling and pipelining our speedup begins to increase significantly (Graph 1). Due to Amdahl's law, given that our arithmetic intensity increases with number of iterations, we see our speed up also increase (Graph 2).

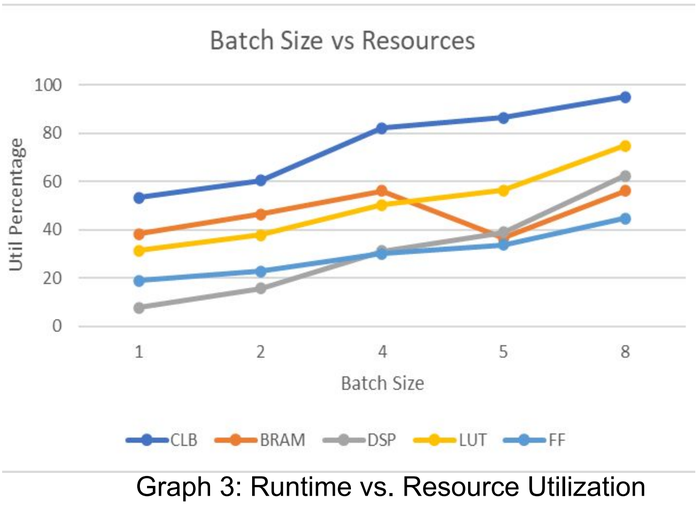


Graph 1: Performance increases from optimizations



Graph 2: Performance vs Iterations

Furthermore, as we increase our batch size, we use more of the on-board resources. Broadly speaking, the FPGA has "storage" resources (Flip Flops and BRAMs) and "compute" resources (Lookup Tables and Digital Signal Processors), and as we hit higher speedup, we use up more of these resources, in this case we are limited by CLBs. Still, unlike traditional accelerators, FPGA resources solely exist for *us* to use, so we do not make as significant of an area tradeoff when we use up more of the board.



Graph 3: Runtime vs. Resource Utilization