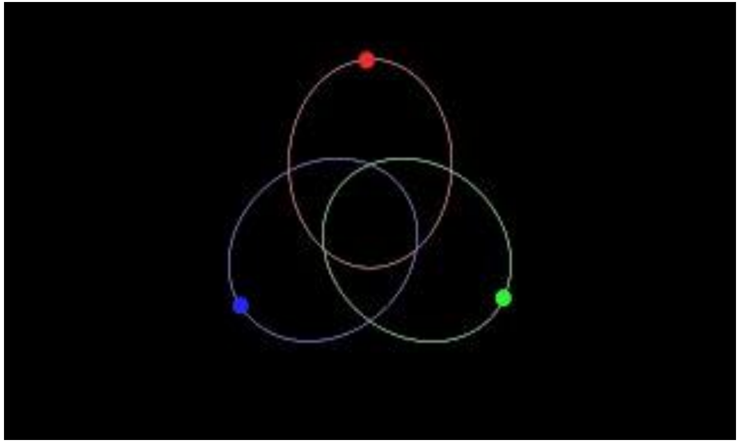
Team A3: N-Body

By Abhinav Gupta, Rene Ravanan, and Yuhe Zheng

Use Case

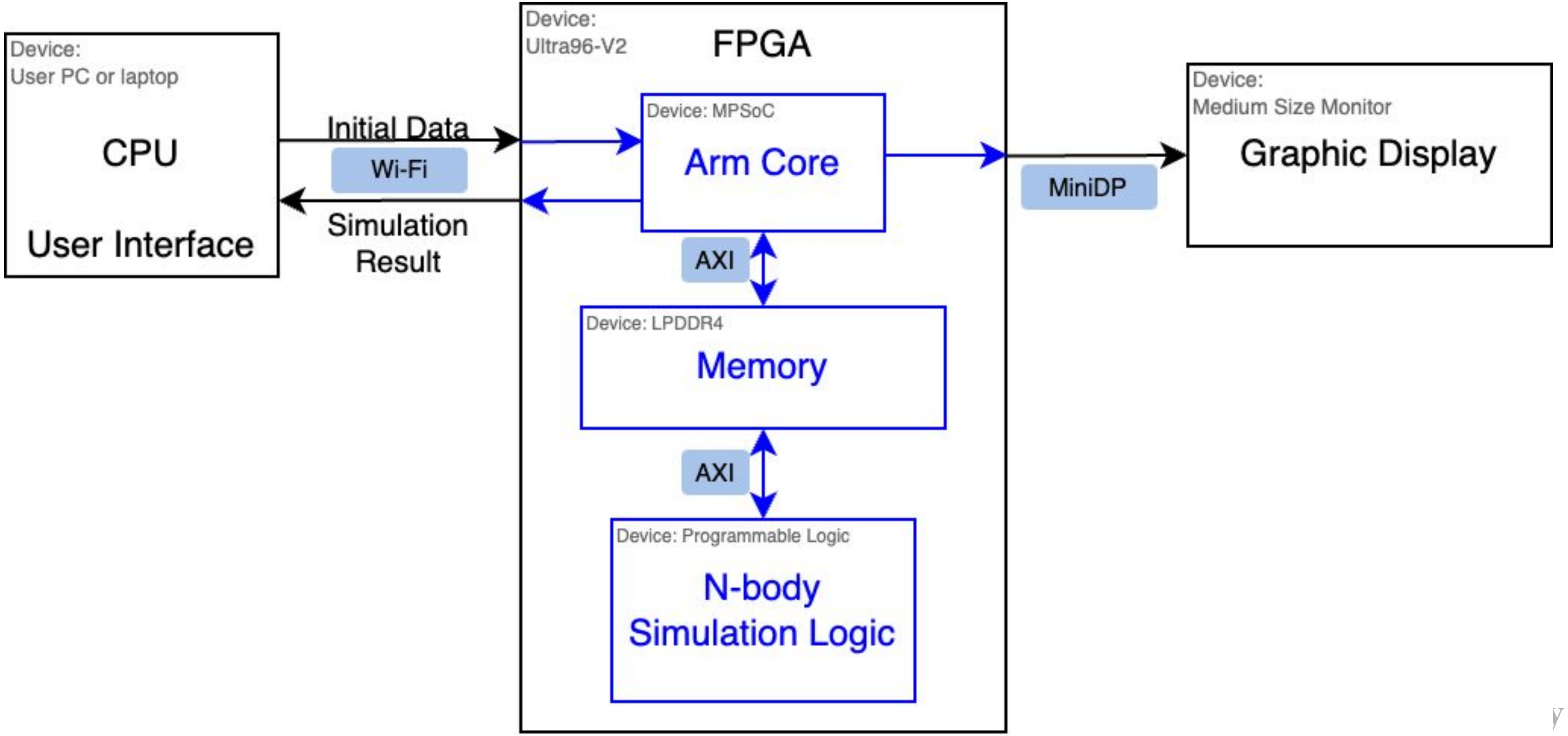
- **Problem:** For physicists the N Body simulation is an important and computationally hard problem to solve, trying to run the algorithm parallely on a CPU is simply not fast enough, and running on GPU is not power efficient
- **Solution:** Run the N Body simulation on a FPGA and try to achieve a **10x speedup**



Quantitative Requirements

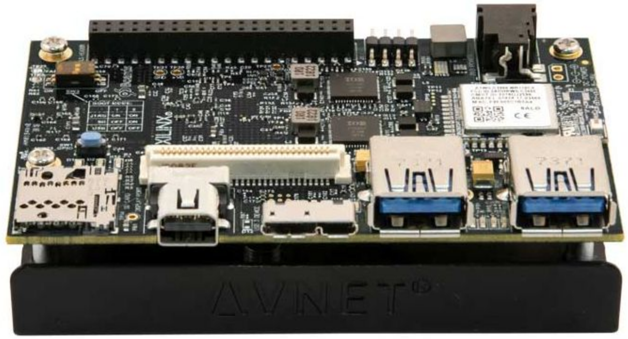
- Our goal is to have a **10x Speedup** for a **~10000 particle** 2D-simulation.
  - **Simulation Size Motivation:**
    - For the use cases that we are considering, (primarily molecular and astronomical simulations) we would need a simulation of roughly 10000 particles.
    - Our FPGA has roughly 70k LUTs, after some research and experimentation we found that this hardware bound matches our use case while making full use of our available resources.
  - **Speedup Motivation**:
    - Ultra96v2 Fabric Clock ~200MHz, ~15x slower than the i7-9700
    - Much of the compute task is data movement
    - Cache/DRAM -> 10s-1000s of cycles; SRAM -> 1s of cycles
    - Multiplying these factors together gets ~10x speedup

Solution Approach and System Overview



Implementation Plan

- **Vitis Platform:**
  - Host Program Compilation
  - High Level Synthesis
  - Utilization Report Collection

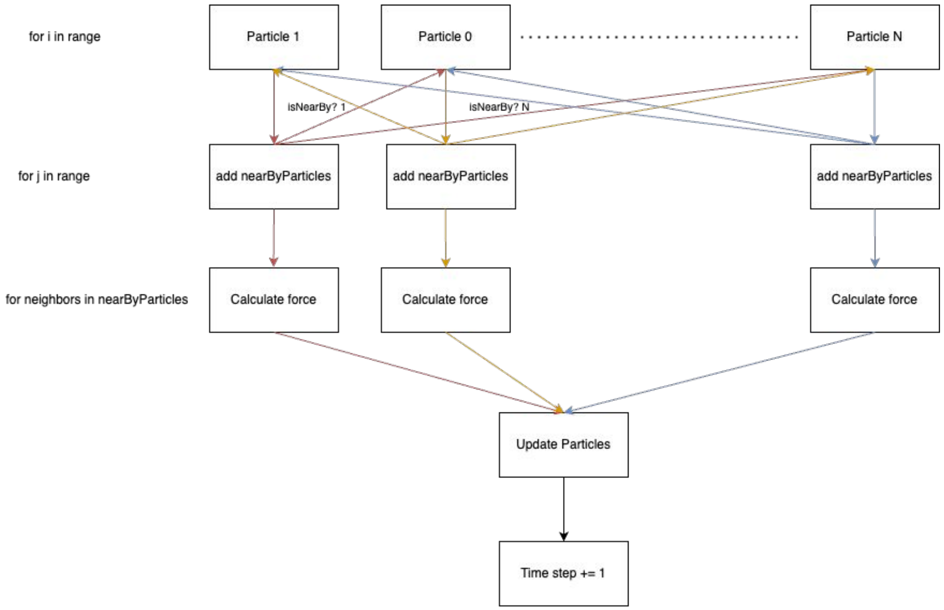


- **Ultra96-V2 FPGA:**
  - Arm Core
  - Programmable Logic
  - MiniDP port for visualization
  - Wifi for data upload

XILINX VITIS™

N-Body Simulations (All pairs)

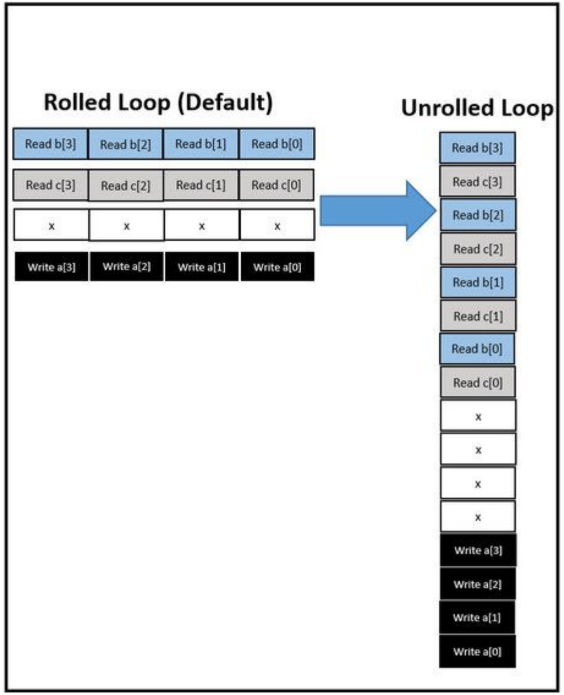
```python
# Main simulation loop
while not simulation_finished:
    # Initialize forces on each particle to zero
    ClearForces()

    # Calculate forces between all pairs of particles
    for i in range(num_particles):
        near_by_particles = []
        for j in range(num_particles):
            if i != j and isNearBy(particle[i], particle[j]):
                near_by_particles.append(particle[j])
        for neighbour in near_by_particles:
            CalculateForceBetween(particle[i], neighbour)
    # Update particle positions and velocities based on forces
    UpdateParticles()

    # Advance simulation time
    UpdateSimulationTime()
```



Algorithm Optimisations

Unrolling:

- Take full advantage of hardware for concurrency
- Run each particle's iteration in parallel



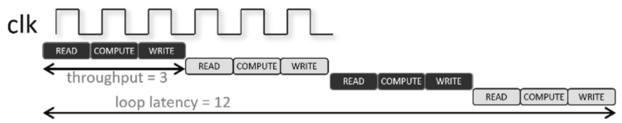

```
# Calculate forces between all pairs of particles
for i in range(num_particles):
    near_by_particles = []
```

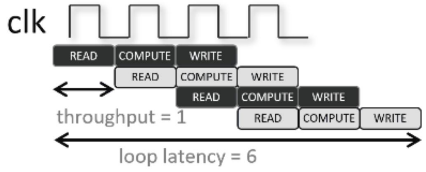
Programmable Hardware Kernel Optimization

Pipelining:

- The PIPELINE pragma enables us to optimise our sections of our code if they are not entirely independent.

```
near_by_particles = []
for j in range(num_particles):
    if i != j and isNearBy(particle[i], particle[j]):  # Exclude self-interaction
        near_by_particles.append(particle[j])
for neighbour in near_by_particles:
    CalculateForceBetween(particle[i], neighbour)
```





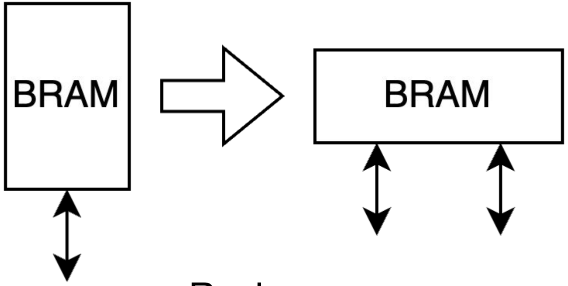
Memory Optimization

Using Block RAM: reconfigurable memory structure customized for each computation

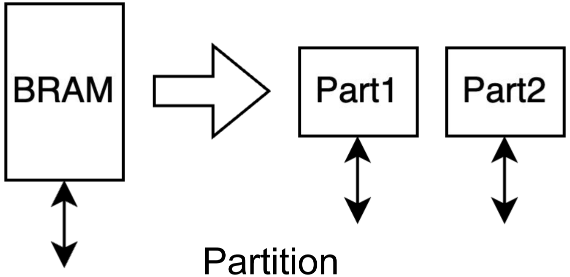
- **Buffer**: store local copy of data to increase DRAM throughput and facilitate data reuse

- **Reshape**: widen memory port to increase bandwidth on consecutive locations accesses

- **Partition**: map one array to multiple BRAMs to allow concurrent computation on multiple elements of array



Reshape

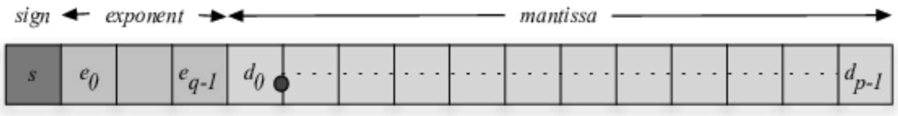


Partition

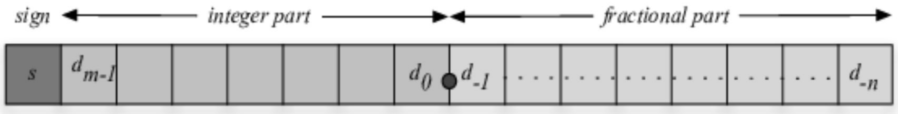
Data Optimisations

Fixed Point Numbers:

- Using fixed point number over floating point numbers takes up less hardware for storage and computation.
- This allows for more hardware to be used for concurrency.



Floating-Point Format

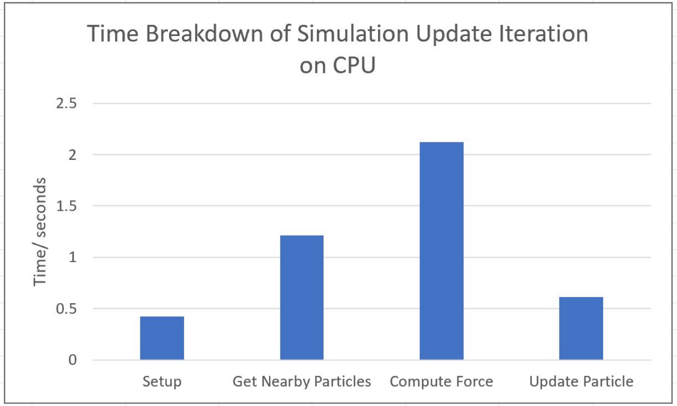


Fixed-Point Format

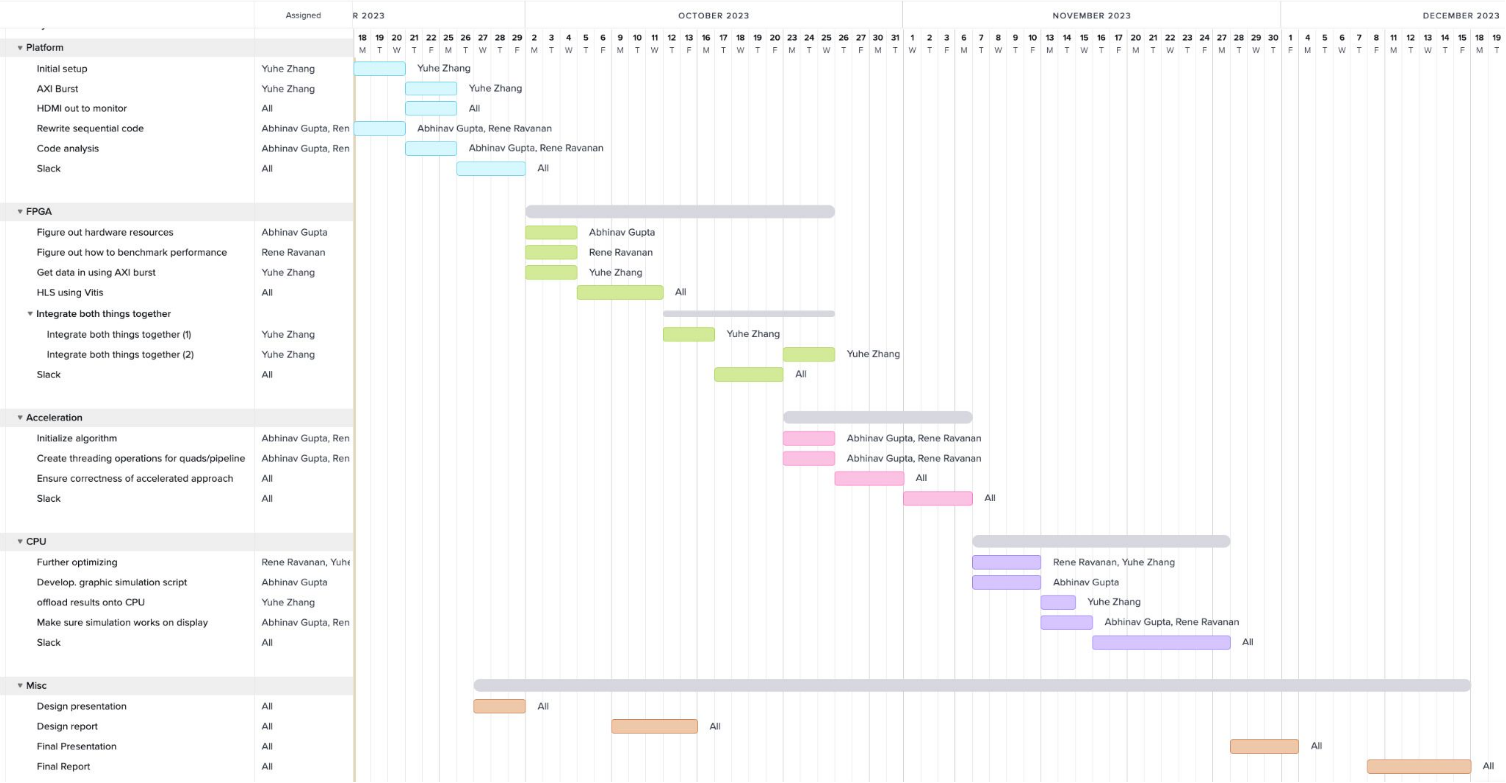
Testing & Verification

Quantitative Verification:

- Correctness: For our use cases (Molecular, Astronomical etc.) a **90-95%** accuracy with our reference solutions would be sufficient for our use cases.
- We aim to focus on optimising the get particles and compute force sections of our simulation.
- Verify our power consumption from the Vitis utilisation report
- Verifying if we achieve 10x speedup



Schedule



Assigned
R 2023
OCTOBER 2023
NOVEMBER 2023
DECEMBER 2023
Platform
Initial setup
Yuhe Zhang
AXI Burst
Yuhe Zhang
HDMI out to monitor
All
Rewrite sequential code
Abhinav Gupta, Rene Ravanan
Code analysis
Abhinav Gupta, Rene Ravanan
Slack
All
FPGA
Figure out hardware resources
Abhinav Gupta
Figure out how to benchmark performance
Rene Ravanan
Get data in using AXI burst
Yuhe Zhang
HLS using Vitis
All
Integrate both things together
Integrate both things together (1)
Yuhe Zhang
Integrate both things together (2)
Yuhe Zhang
Slack
All
Acceleration
Initialize algorithm
Abhinav Gupta, Rene Ravanan
Create threading operations for quads/pipeline
Abhinav Gupta, Rene Ravanan
Ensure correctness of accelerated approach
All
Slack
All
CPU
Further optimizing
Rene Ravanan, Yuhe Zhang
Develop. graphic simulation script
Abhinav Gupta
offload results onto CPU
Yuhe Zhang
Make sure simulation works on display
Abhinav Gupta, Rene Ravanan
Slack
All
Misc
Design presentation
All
Design report
All
Final Presentation
All
Final Report
All

Recap

MVP

- Goal is to make the 2D N-body simulation run **10x faster** on the FPGA. **N = 10k**, **time-steps = 100k**

How are we going to achieve it

- Use algorithmic optimization (**Unrolling**)
- hardware kernel optimization (**Pipelining**)
- Memory Optimization (**Block RAM**)
- data optimization (**fixed point numbers**).

How are we going to verify it

- Achieve a **90-95%** accuracy when compared to the reference solution