FPGA Accelerated N-body Simulations

Authors: Abhinav Gupta, Rene Ravanan, Yuhe Zheng
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— A system aiming to leverage the efficient programmable hardware of an FPGA to accelerate N-body simulations. N-body simulations play a pivotal role in astrophysics, molecular dynamics, and a wide range of scientific fields. The report highlights the limitations of traditional CPU and GPU based approaches, emphasizing the pressing need for more efficient solutions. By leveraging FPGAs, the project aims to achieve substantial speedup and simultaneously reduces power consumption, making it a cost-effective solution.**

***Index Terms*—Acceleration, BRAM, FPGA, N-body Simulations, Newton's law of Gravitation, Pipelining, Ultra96, Vitis**

1 INTRODUCTION

N-body simulations are fundamental tools in the realm of computational physics, used to model the interactions between multiple particles or bodies within a dynamic system. These simulations play a crucial role in understanding various complex phenomena, including gravitational forces among celestial bodies in astrophysics, the behavior of molecules in molecular dynamics, and even the dynamics of particles in simulations of fluid flow. The primary challenge in N-body simulations lies in the computation of the gravitational forces or other interactions between each pair of particles, which scales quadratically with the number of particles, making it a computationally intensive task.

Traditional approaches to N-body simulations often involve the use of Central Processing Units (CPUs) or Graphics Processing Units (GPUs). While these methods are valuable, they often face limitations in terms of speed and power efficiency, particularly when dealing with simulations that involve a large number of particles which have very irregular computation patterns. To address these limitations, this paper proposes a pragmatic solution: leveraging FPGAs to enhance the performance of N-body simulations.

FPGAs are reconfigurable hardware devices that offer the potential to significantly enhance the performance of N-body simulations while simultaneously addressing power consumption concerns. The driving motivation behind this project stems from the practical need to advance computational physics. Our central goal is to run N-body simulations involving a substantial 10,000 particles, striving to attain a tenfold speedup compared to CPU-based methods.

Our project aims to incorporate FPGA acceleration into computational physics, with a focus on practical applications. This approach is intended to improve computational efficiency, particularly for graduate students and researchers with limited resources. By doing so, we aim to facilitate the simulation of systems involving a substantial number of interacting particles, which is often a costly and resource-intensive task. This practical development seeks to support the scientific community, especially in fields like celestial mechanics and molecular dynamics, by offering a more cost-effective solution for conducting research without the need for excessive claims or resource investments. By providing a solution that is more power efficient than its GPU counterparts, we also aim to reduce our negative impact on the environment as well.

2 USE-CASE REQUIREMENTS

In framing our use case for the FPGA-based N Body simulation system, we've established a multifaceted set of objectives that align with our goals. Our primary aim is to achieve a substantial speedup compared to traditional CPU-based simulations, thereby significantly improving computational efficiency and reducing research time.

Incorporating public health, safety, and welfare considerations, our FPGA system will prioritize user safety through robust safety measures. Potential hazards and risks associated with FPGA technology, including overheating concerns, are addressed comprehensively, ensuring a secure research environment. Beyond safety, our approach emphasizes accessibility. The user-friendly design, including a display interface, is developed with sensitivity to diverse backgrounds, promoting a collaborative and globally accessible research environment.

In addition, our commitment extends to environmental sustainability by minimizing power consumption, aligning with environmental goals for responsible high-performance computing. Simultaneously, economic factors play a crucial role in ensuring affordability and accessibility, particularly for students and researchers with limited resources, fostering economic inclusivity within the research community. This approach blends speed, safety, inclusivity, environmental responsibility, and economics to create a comprehensive N Body simulation solution.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

We implemented our solution with a straightforward architecture. We have a host computer sending the initial simulation conditions (positions of the particles, their mass, and velocity) to our acceleration hardware. Our acceleration piece of hardware that we plan to use the Xilinx Ul-

traScale+ Ultra96v2 development board. This board also has an ARM core integrated into facilitating the running of C++ and flexible FPGA fabric for accelerated kernels. We plan to have the entire simulation run on the FPGA with the results being displayed both on a monitor and sent back to the user.

Our high-level approach is to have the FPGA's ARM core manage the launching of the simulation kernels, including transferring the particle data to and from the FGPA fabric and FPGA memory. We would transfer data between our host computer and FPGA be secure copying (scp) the files to/from the ARM core. The output files would have a resultant position and velocity of the particles at each time step. Once the specified number of iterations for the simulation, this will be sent back to the user, as mentioned above, and also be displayed on a display connected to the FPGA via its mini DP port as a visual reference. Figure 1 shows a block diagram representing this.

The main algorithm for our simulation can be split into four discrete steps where our optimisations will take place. These steps consist of arithmetic computation such as multiplication, addition, dot products etc. This will also comprise of data transfer between some data structures that store physical information.

We will be writing our code in C++ using Vitis HLS to synthesize our logic onto the FPGA fabric. Doing so allows us to take advantage of the FPGA's performant infrastructure while still maintaining the ease and familiarity of writing code for a CPU. This would allow us to focus primarily on the actual optimisation of our design and algorithm and not on getting our code to run on the FPGA.

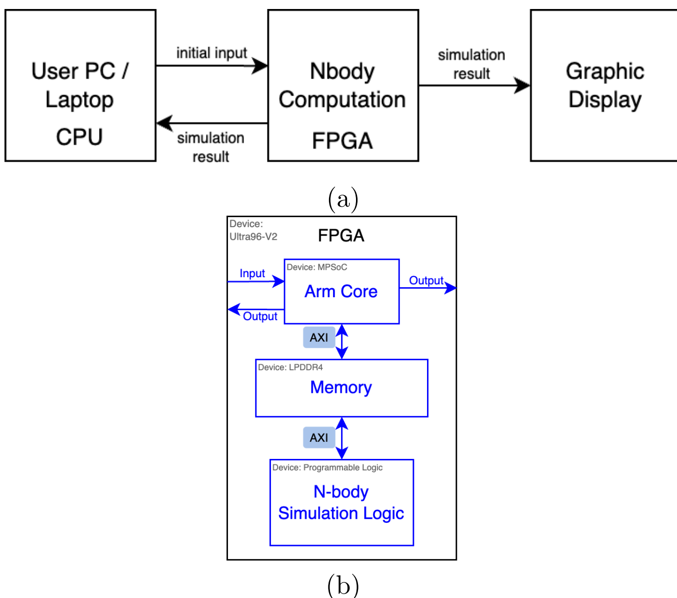

Figure 1: System Description (a) overall system (b) inside FGPA

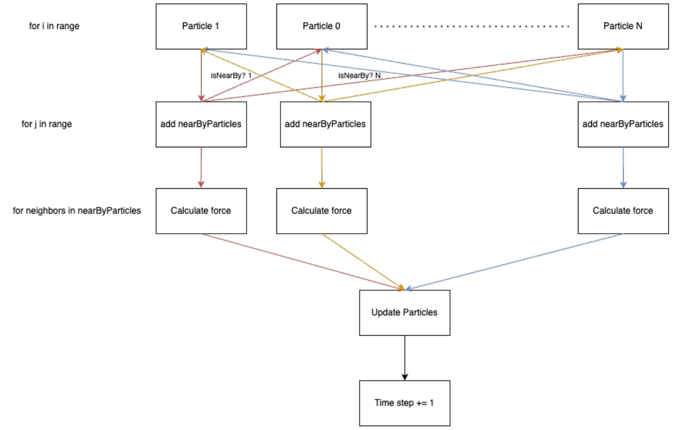

Figure 2: Data flow diagram describing all pairs alogirthm

```
# 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()
```

Figure 3: All pairs algorithm code

Fig. 3 above describes the all pairs algorithm which we will be parallelizing. The as we can see the algorithm runs in $O(N^2)$ because the outer most loop loops through all the particles $1...N$, the inner loop also goes through all the particles and determines which particles are nearby to the particles $i \in 1...N$. This is after all the nearby particles have been determined we then calculate the force on particle i and then update the position of the particle. After this is done for all the particles $1...N$ then we increase our time by 1.

4 DESIGN REQUIREMENTS

4.1 Speedup

The 10x speedup is achieved by optimizing the computational aspects of the simulation. This is our most important requirement as this is the speedup our physicists need in order to make our product viable for them. To motivation behind our speedup goal was to see what was theoretically the max speedup we could achieve on an FPGA. The fabric clock on an FPGA runs at around 200MHz

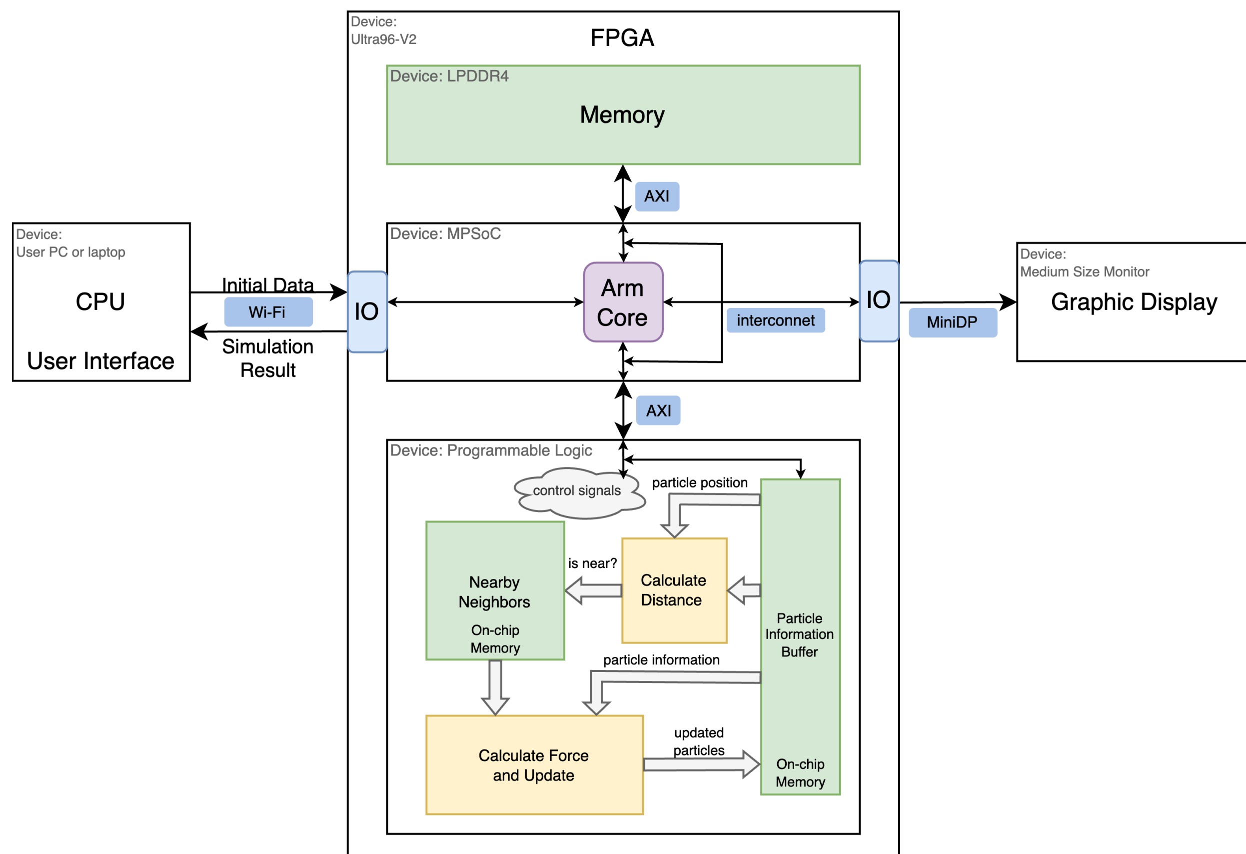

Figure 4: Detailed Block Diagram

which is about 15x slower than a i7-9700 (this is the CPU we are running our reference calculations on). But we have seen than the Cache/DRAM on the FPGA is significantly faster than the CPU. The memory is somewhere near 10x-100x faster. If we take both of these factors into consideration then we can see a theoretically viable and achievable speedup is around 10x.

4.2 Simulation Scale

The system should support simulations with a minimum of 10,000 particles in a 2D environment. The scale should be suitable for molecular and astronomical simulations, ensuring meaningful results. [4] The way we arrived at this simulation was estimating our hardware resources, on our FPGA we have 70K LUTS which should be able to support data transfer and calculation of around 10000 particles [2]. This number does remain flexible though in case further testing reveals to us that we might be more hardware bound.

A 10k particle simulation before might have taken around 20hours to run. We are very happy with our speedup because it will be able to cut down the simulation time to merely 2hours. This speedup is crucial in stratifying our use case which is first and foremost being able to be helping the under resourced physicists out there. This also helps us achieve our other environmental and collaborative goals that in order to achieve a 10x speedup only an FPGA will be needed and no extra components, this will help cut down costs, global waste and will encourage more cross platform partnerships within academia.

5 DESIGN TRADE STUDIES

For our project, we had to make several considerations and decisions when approaching our problem: Accelerating N-body simulations in an efficient and cost-effective manner. To start with we first had to choose our acceleration platform, our FPGA, then our simulation algorithm and finally our hardware acceleration approaches.

5.1 Hardware Platform

As mentioned above and in our introduction, iterations of the N-body simulation already exist on CPUs, GPUs and other platforms. When choosing our platform we had to carefully evaluate the costs and benefits of each of them, how they can cope with our workload, their efficiency and accessibility. The three main platforms we considered were a CPU, GPU, ASIC and an FPGA. Below are the pros and cons of each that we considered.

5.1.1 CPU with Multithreading

A comparatively straightforward approach to this problem would be to use a CPU to conduct our simulation. Two of us have already tried this approach in the class 15-418: Parallel Computer Architecture and Programming. The intuitive way to parallelise this simulation is to do so on the axis of each particle during the compute force section of our simulation. This would not be the most efficient on a CPU given that we are dealing with around ten thousand particles at a time and most standard CPUs allow 8-16 threads (with super computers using 128). Additionally, thread synchronization and communication costs would be very high given that we would have many shared data structures. Additionally, CPU kernels would also spend time context switching between different applications and processes that are running devoting lesser time to our simulations and making running them all the more expensive.

5.1.2 GPU

GPUs solve the above mentioned problem of CPUs being hardware bound by the available concurrency on the platform as each block on a GPU can spawn up 1024 threads making this very scalable for our use case. However, our computation can be very irregular, for example particles that are grouped together would have to perform larger resultant force computations as compared to sparsely located ones. This would again lead to several overhead communication costs between threads and GPU blocks which significantly hurts our performance. Moreover, the memory architecture of a GPU would force us to use its global memory instead of its shared block memory given that all blocks need to have accesss to the particle information. Reading/Writing data in global memory of a GPU is also very expensive. Lastly, a GPU is not very power efficient, an average GPU has a TDP of 800W. Even though some fast implementations of our problem have been designed on GPUs, their adverse effects on the environment with their power consumption makes us want to work on a more sustainable platform.

5.1.3 ASIC

ASIC is an efficient approach, however, given the time constraints that we are presented with, we would be able to design and fabricate a chip. Our design would also have to be produced at scale for this to be worthwhile, making this not ideal for a prototype system.

5.1.4 Why FPGA?

Given the above constraints with other platforms, we chose the FPGA as our target platform. FPGAs allow us to take advantage of customisable hardware to exploit parallelism in conducting our N-body Simulation. Given that our computation is irregular, we can customise our hardware to also deal with this, i.e. instantiate extra hardware for cases that require it such as when particles are clustered together. FPGAs also offer high flexibility in their memory architecture with BRAM allowing us to take advantage of memory reuse with out the costly high latency trips of DRAM. We have also found well-documented tooling available for accelerating code on FPGAs including papers that have tried our N-body simulations on cloud FPGAs. FPGA kernels would also not have to deal with as much overhead as a CPU kernel as the former would be focusing solely on our computation. Lastly, compared to options like ASIC they are also relatively cheaper and is also more power efficient than GPUs as mentioned in our design requirements.

5.2 Hardware Acceleration Approach

5.2.1 HLS vs. HDL

After choosing the FPGA as our platform, we needed to decide how we were going to write our kernel code. This could have either been done using a Hardware Description Language (HDL) like System Verilog, or using a more programmer friendly language like C++ and use High Level Synthesis to synthesise this onto the FPGA. The first consideration we had was the our initial CPU implementation was already written in C++, choosing to with HLS would mean that we could spend more time optimising a familiar and approachable algorithm instead of spending time getting an initial implementation setup and running. Additionally, HLS also offers several tools to facilitate parallelism, such as pipelining Pragmas on Vitis HLS, and that the compiler also is able to infer and parallelise independent sections of our code which programming in SystemVerilog would not allow.

5.2.2 Fixed-Point vs Floating Point Numbers

Another consideration that we had on accelerating our design on the hardware side was the choice between fixed-point and floating point numbers. Floating point numbers do offer higher precision, but they are more expensive in computations and also take up more hardware. Given our accuracy requirement of 90-95% as specified in our design and testing sections, using 16.16 fixed point numbers would be sufficient for this while giving us more hardware to exploit for concurrency.

5.3 N-body Simulation Algorithms

After finalising our hardware platform and acceleration approach, we had to choose the algorithm that we would run our simulations with. There are several proposed alrgorithms to run N-body simulations, the two most widely accepted ones are the Barne's Hut and All Pairs algorithms. Their tradeoffs have been discussed below

5.3.1 Barne's Hut Algorithm

The Barne's Hut algorithm has an $O(NLogN)$ time complexity where N is the number of particles. Here at each time-step, a particle can find its nearby particles to

compute its resultant force and velocity vectors using a data structure called a quad tree. This data structure, as seen in the figure **fill this**, splits the simulation space into quadrants where each node represents one quadrant and continues to recursively do so until each leaf has only 1 particle or has child nodes that split it further. Traversing this to find nearby particles is considerably more efficient ($O(LogN)$ vs $O(N)$). However, implementing this in HLS would be quite complex and reading/updating this data structure would be quite expensive on an FPGA.

5.3.2 All Pairs Approach

This approach is slightly less sophisticated, at each time-step, each particle loops over every other particle to find out if it is nearby and then compute its interactive forces. This approach as a higher time complexity of $O(N^2)$. However, this is a lot more easy to parallelise given its comparitive independence in force computations and update phases. It is also easier to store into memory on an FPGA making this a more widely used algorithm in accelerating N-body simulations in platforms such an FPGA or a GPU. Hence, given its comparative ease to optimise, we chose the All Pairs approach.

6 SYSTEM IMPLEMENTATION

As mentioned in the architecture section, our hardware platform will be the Xilinx Ultra96v2 development board. A user can interface with this board to send/receive data from it via wifi, by copying input and result files to/from the board via scp and ssh. The graphical display will be connected to the board via its mini DP port.

With respect to our actual optimisations, we have mentioned our plan below.

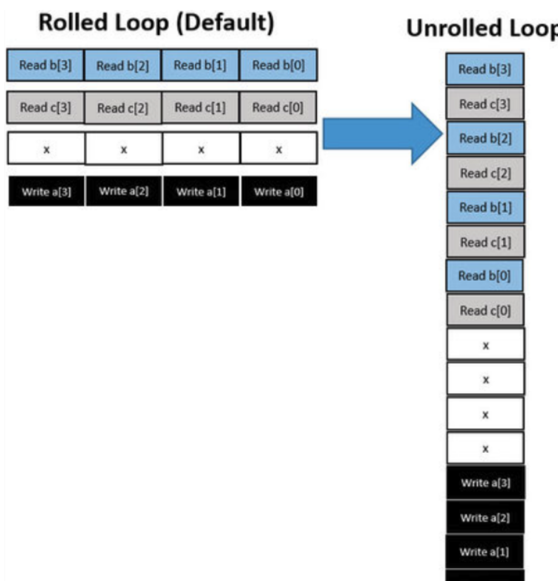

Figure 5: Rolled vs Unrolled loop [5]

6.1 Loop Unrolling

Loop unrolling is a critical optimization technique employed in parallel computing to improve the execution speed of iterative loops within algorithms. In the N-body simulation, loop unrolling involves unwrapping a loop, essentially transforming a series of repetitive iterations into a sequence of independent and parallelized operations. Instead of processing a single element or body per iteration, loop unrolling allows for the simultaneous handling of multiple elements within a single iteration, making more efficient use of the hardware's capabilities and increasing computational throughput.

In the "all pairs" algorithm, loop unrolling has a substantial impact on its performance. In the N-body simulation, each body interacts with every other body to calculate gravitational forces. The "all pairs" algorithm has nested loops for pairwise interactions. By unrolling these loops, we can effectively perform multiple pairwise interactions within a single iteration, reducing the overhead of loop control and enabling the utilization of vectorized instructions or parallel hardware. This optimization technique significantly accelerates the simulation by decreasing the number of loop iterations and improving the overall runtime efficiency. Loop unrolling, therefore, proves instrumental in enhancing the speed and efficiency of N-body simulations, making them suitable for tackling complex problems in fields like astrophysics, molecular dynamics, and computational chemistry.

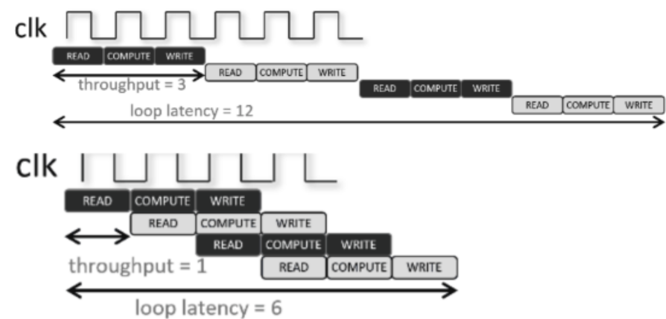

Figure 6: Unpipelined vs Pipelined [6]

6.2 Pipelining

Pipelining is an optimization technique in computing used to improve the throughput of processing elements by breaking down tasks into smaller, sequential stages. When applied in the context of hardware acceleration using Vitis, a pragma for pipelining defines how an operation or set of operations should be divided into stages for concurrent execution. In essence, it enables parallel processing of multiple elements or tasks in a streaming fashion, reducing latency and increasing throughput.

In the N-body simulation, each body interacts with every other body to calculate gravitational forces or other interactions. With the pipelining pragma, these interactions

can be divided into stages, such as data fetching, computation, and result storage. The pragma enables overlapping the execution of different stages, making more efficient use of hardware resources. For instance, while one stage is processing data for a specific interaction, the next stage can start fetching data for the subsequent interaction. This concurrent processing reduces overall execution time, making the simulation run faster and efficiently utilizing the FPGA's parallel processing capabilities. In essence, pipelining is a valuable technique for optimizing the N-body simulation on FPGAs, allowing it to tackle large-scale simulations with improved throughput and reduced time to calculate results.

6.3 FPGA Memory

A major advantage of computation on FPGA is that we can implement custom memory structure to fit the algorithm's data accessing pattern. This way, we can improve the memory throughput which usually bottlenecks memory-intensive computations.

6.3.1 DRAM

The DRAM on the FPGA stores the initialization and simulation results of our system. The host program in the Arm core stores the initial state of the particles received from the user's PC or laptop to the DRAM before the simulation starts. During simulation, the N-body computation hardware kernel in the programmable logic continuously reads the particles' information - position, velocity, and mass - from DRAM. The kernel calculates the particles and then writes back the simulated results. At the end of the simulation, the FPGA extracts the simulated results into files for the user to retrieve from the board. The DRAM of the Ultra96-V2 FPGA contains 2GB memory and has a measured read latency of around 500ns - several dozens of cycles depending on the computation's target frequency[2]. This long latency means we need to perform our computation on locally buffered data instead of making frequent trips to DRAM. We also need to batch-read data from DRAM to local buffers to maximize DRAM throughput.

6.3.2 LUTRAM

LUTRAM is on-chip memory implemented from LUTs in the programmable fabric. LUTRAM can have 32-bit or 64-bit wide ports, and it supports dual-edge clocking for fast data access [1]. Therefore, LUTRAM is ideal for fast and flexible data access. However, because LUTRAM is implemented using LUTs, more data stored in LUTRAM means less LUTs can be used for calculation. Therefore, we should only store data structures that need to be accessed in every cycle in LUTRAM to optimize resource usage.

6.3.3 Block RAM

Block RAM is a hardened fast memory structure in the programmable fabric. Block RAM supports dual port access, which means we can read and write from Block RAM at the same time. Block RAM ports are reconfigurable. It supports reshaping and partitioning memory. This means we can access multiple elements of an array at the same time, enabling us to parallelize a computation by pipelining and unrolling loops. Therefore, we should store the majority of our data in Block RAM for efficient use of memory resources.

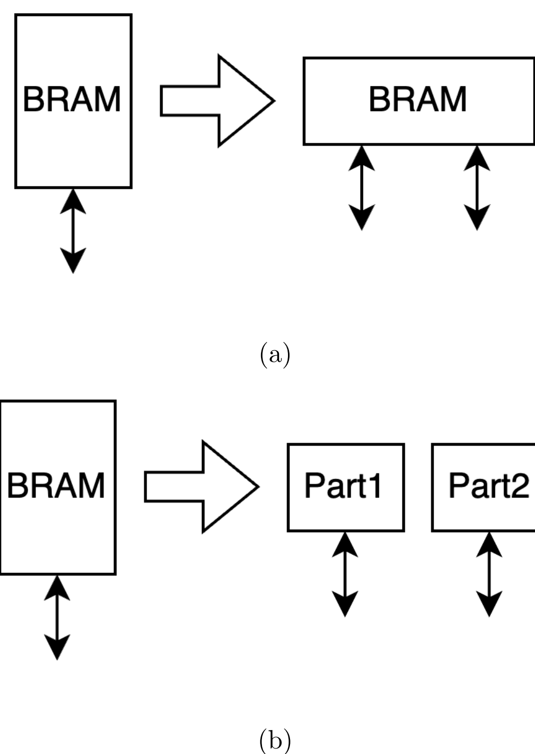

Figure 7: BRAM Configuration (a) Array Reshape (b) Array Partition

7 TEST & VALIDATION

7.1 Accuracy Tests

For our given use cases of Molecular and Astronomical N-body Simulations, we found that a $90 - 95\%$ accuracy would be sufficient. Hence, the first thing that we will check with our project is whether the N-body simulation results that we produce satisfy the requirements in accuracy. To do so, we have a few reference simulation results that we can run to verify this. We also have our benchmark CPU implementation which we developed in a previous class as another method of validation.

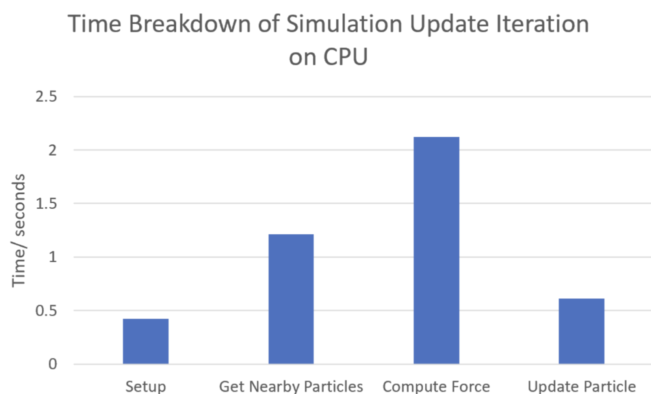

Figure 8: Timing break down of CPU code

7.2 Verifying Speedup

From our code snippet above in Fig. 8 and from the data from our CPU benchmark, we can see that the get particles and compute force sections of our code takes the most time and we therefore plan to achieve our speedup by optimising this section of our code first. We will verify this by timing our FPGA computation and comparing this with the CPU benchmark that we have found.

7.3 Power Consumption

Given that we are trying to also be energy efficient, we will have to verify that running our code on an FPGA does not exceed that of a CPU or GPU. We will confirm this using the Vitis utilisation report. For a sanity check, since we also know the power requirements of our Ultr96v2 board from its data-sheet, we can roughly estimate its power consumption using this information and the time our code took to run.

8 PROJECT MANAGEMENT

8.1 Schedule

Refer to our Gantt chart for our schedule. We have accounted for slack time and breaks as well.

8.2 Team Member Responsibilities

Initially Abhinav and Rene worked on setting up the CPU benchmark and Yuhe worked on setting up the Ultra96 FPGA. Abhinav and Rene would then focus on algorithm analysis and learning about Vitis HLS optimisations while Yuhe setup our Vitis HLS workspaces. Towards the latter half of our project once we all would focus on achieving our speedup goals. Abhinav would also spend some time writing a script that runs on the ARM core to produce a visual representation of our simulation results.

8.3 Bill of Materials and Budget

Since Yuhe is taking 18-643 at the time of this project, we were able to borrow her FPGA at no additional cost. Given the software and digital hardware oriented nature of our project, we do not foresee any additional costs. For the graphical display, we would test on our own monitors at home and then borrow one from the ECE clusters for our demonstrations.

8.4 Risk Mitigation Plans

One challenge we can face is running out of the programmable fabric's hardware resources. There are two types of resources we can run out of: computation resources and memory resources. The high-level synthesis compiler uses DSPs for calculation by default, but the Ultra96-V2 FPGA only has 360 DSPs, which limits the amount of computation we can do in parallel. In case we need more computation units, we can intentionally tell the compiler to use LUTs to perform some calculations, providing more computing capabilities. BRAM resources can also run out when we parallelize data access to a degree that uses up all the BRAM ports. To solve this issue, we need to rearrange our data accessing pattern, so that we partition data to fit BRAM block shape for efficient BRAM usage. If both strategies do not work, we need to parallelize less and improve speedup in other ways, such as increasing the frequency of our computation.

9 SUMMARY

To conclude, our project aims to accelerate N-body simulations on an FPGA. N-body simulations are a computationally intensive simulations run by molecular and astronomical physicists and current implementations of this are slow and/or inefficient. We aim to leverage the FPGA's versatility and power efficiency to contribute to making these simulations more accessible to scientists on a budget while being sustainable for the environment. Our FPGA of choice is the Xilinx Ultra96v2, and hope to both accelerate and produce a visual representation of this simulation. Our plan for carrying out this optimisation involves taking advantage of Vitis Pragmas like pipelining, and FPGA memory features like BRAM. While we do foresee complications with working with Vitis and some of the hardware restrictions of the FPGA (LUT count etc.), we are confident that we can come up with an optimal design to meet our design goals. [3]

Glossary of Acronyms

- ASIC - Application-specific Integrated Circuit
- BRAM - Block Random Access Memory
- CPU - Central Processing Unit

- DRAM - Dynamic Random Access Memory
- FPGA - Field Programmable Gate Array
- GPU - Graphics Processing Unit
- HDL - Hardware Description Language
- HLS - High-Level Synthesis
- LUT - Lookup Table
- DSP - Digital Signal Processor
- LUTRAM - Lookup Table Random Access Memory
- SRAM - Static Random Access Memory
- TDP - Thermal Design Power

References

[1] AMD Adaptive Computing Documentation Portal. *Versal ACAP Configurable Logic Block Architecture Manual (AM005)*. Accessed: 14 October 2023. 2023. URL: https://docs.xilinx.com/r/en-US/am005-versal-clb/Look-Up-Table.

[2] Avnet. *Ultra96-V2 Single Board Computer Hardware User's Guide*. Accessed: 15 October 2023. 2021. URL: https://www.avnet.com/wps/wcm/connect/onesite/b85b9556-0b2a-42b3-ad6a-8dcf3eac1ff9/Ultra96-V2-HW-User-Guide-v1_3.pdf?MOD=AJPERES.

[3] Intel® FPGA SDK for OpenCL™ Standard Edition: Best Practices Guide. *Floating-Point versus Fixed-Point Representations*. Accessed: 14 October 2023. n.d. URL: https://www.intel.com/content/www/us/en/docs/programmable/683176/18-1/floating-point-versus-fixed-point-representations.html.

[4] Rajeev Patwari and N.C.K. Choy. *Ultra96 FPGA-Accelerated Parallel N-Particle Gravity Sim*. Accessed: 14 October 2023. Hackster.io. 2019. URL: https://www.hackster.io/rajeev-patwari-ultra96-2019/ultra96-fpga-accelerated-parallel-n-particle-gravity-sim-87f45e.

[5] Murali Ravi et al. "FPGA as a Hardware Accelerator for Computation Intensive Maximum Likelihood Expectation Maximization Medical Image Reconstruction Algorithm". In: *IEEE Access* PP (Aug. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2932647.

[6] wordchao. *HLS Optimization: Throughput*. cnblogs. 2019. URL: https://www.cnblogs.com/wordchao/p/10943227.html.

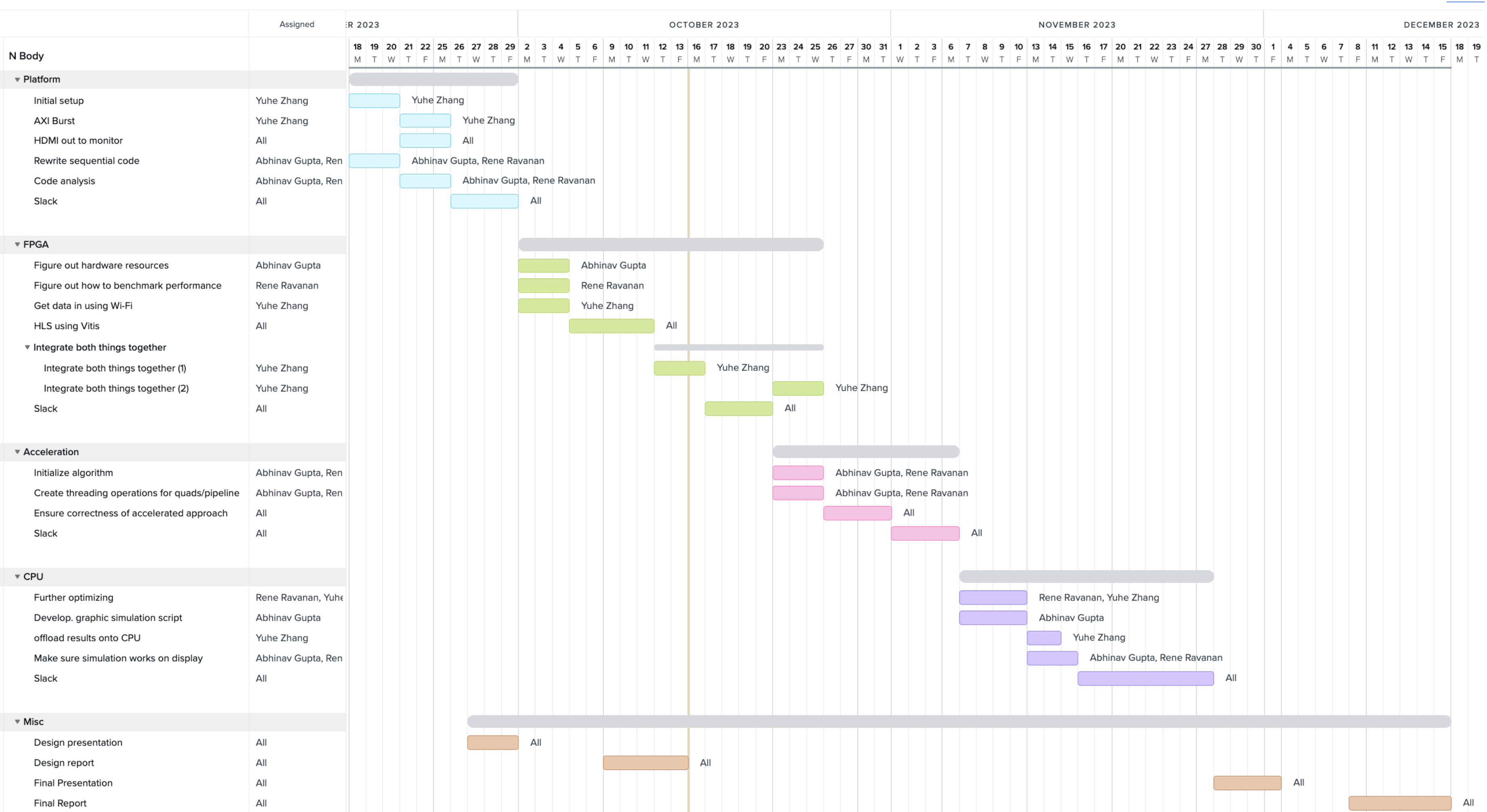

Figure 9: Gantt Chart