# FPGA-Accelerated Fluid Simulation

**Team E1: Jeremy Dropkin, Alice Lai, Ziyi Zuo**
18-500 Capstone Design, Spring 2022
Electrical and Computer Engineering Department
Carnegie Mellon University

## Product Pitch

Fluid simulation is a computer graphics problem uniquely suited for FPGA acceleration due to high workloads and high amounts of irregularity in computation. Our goal is to apply FPGA acceleration to a new approach in fluid simulation: *position-based* fluids. We want to provide at least a **10x speedup** compared to a CPU implementation.

Using Scotty3D, a custom 3D graphics software package, we developed an FPGA-based fluid simulation system that outputs simulation log files which can be played back in Scotty3D offline. We were able to generate visually and mathematically accurate fluids with a speedup of **50x** over the CPU implementation.

This product is suitable for 3D animation artists who would like to simulate a lot of different fluid simulation scenes very quickly.

## System Architecture

Our system operates both on the CPU and fabric (reconfigurable logic portion) of the FPGA board. The CPU takes care of managing the file IO and managing user scenes. It then sends instructions to the FPGA to begin the fluid simulation by transferring data via the highly efficient AXI Burst protocol (commonly used on many modern SoCs).
The output is sent back to the CPU from the FPGA.

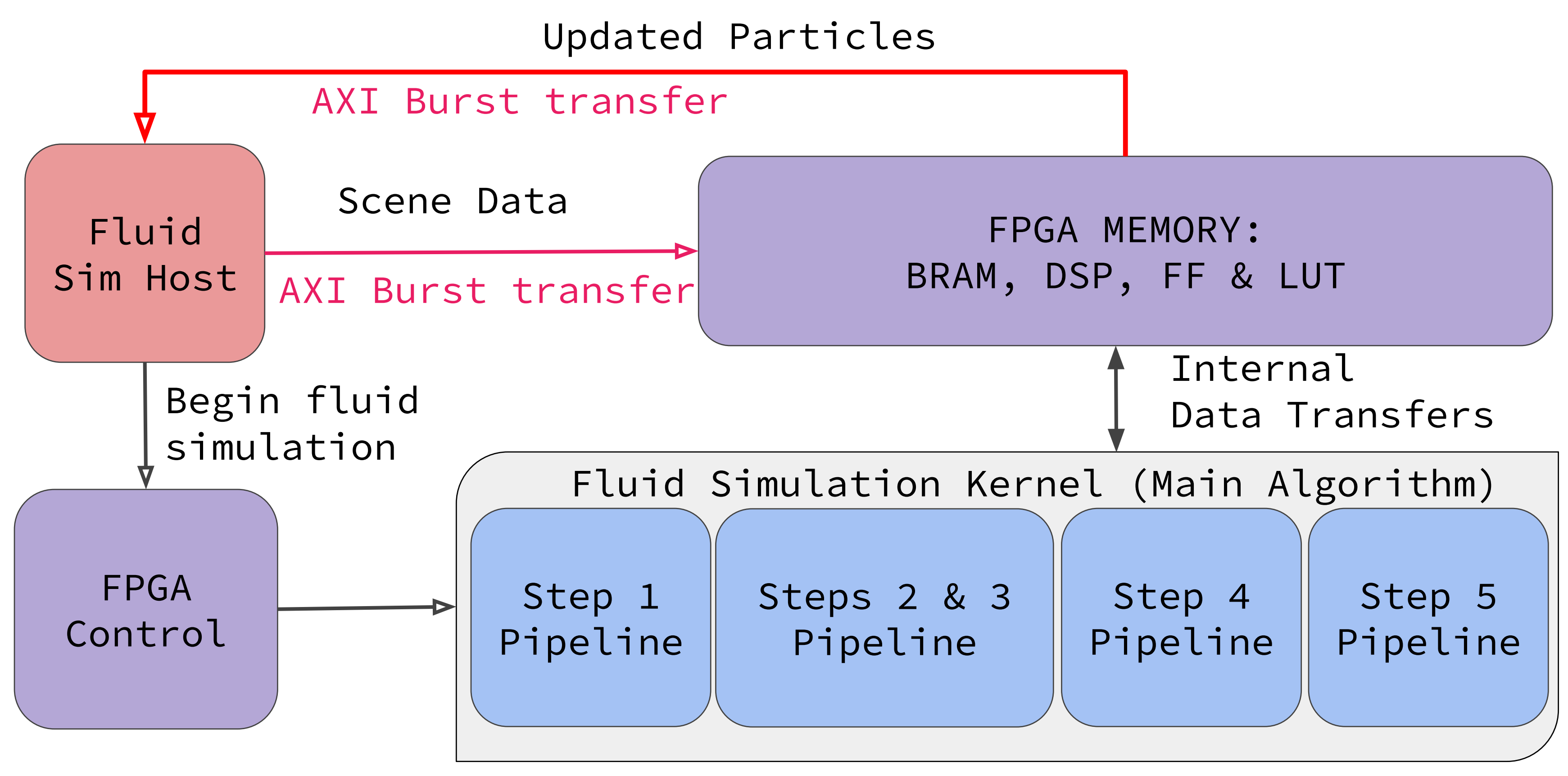


Figure 1: System Block Diagram

| Optimization | Description |
|---|---|
| Pipelining | Increase hardware utilization by having different sets of inputs executing different stages of the algorithm concurrently, rather than waiting for one input to finish all steps |
| Unrolling | Create more hardware in order to execute independent loop iterations in parallel so execution time decreases. |
| Fixed-Point Numbers | Increase speed of mathematical operations and use less space to store numbers with a tradeoff of lower precision |
| Bounded 3D voxel space | Ensure O(1) lookup time for neighboring particles (a very frequent operation) |

Table 1: Optimizations Summary

## Conclusions & Additional Information

Scan to get read project details on our website.

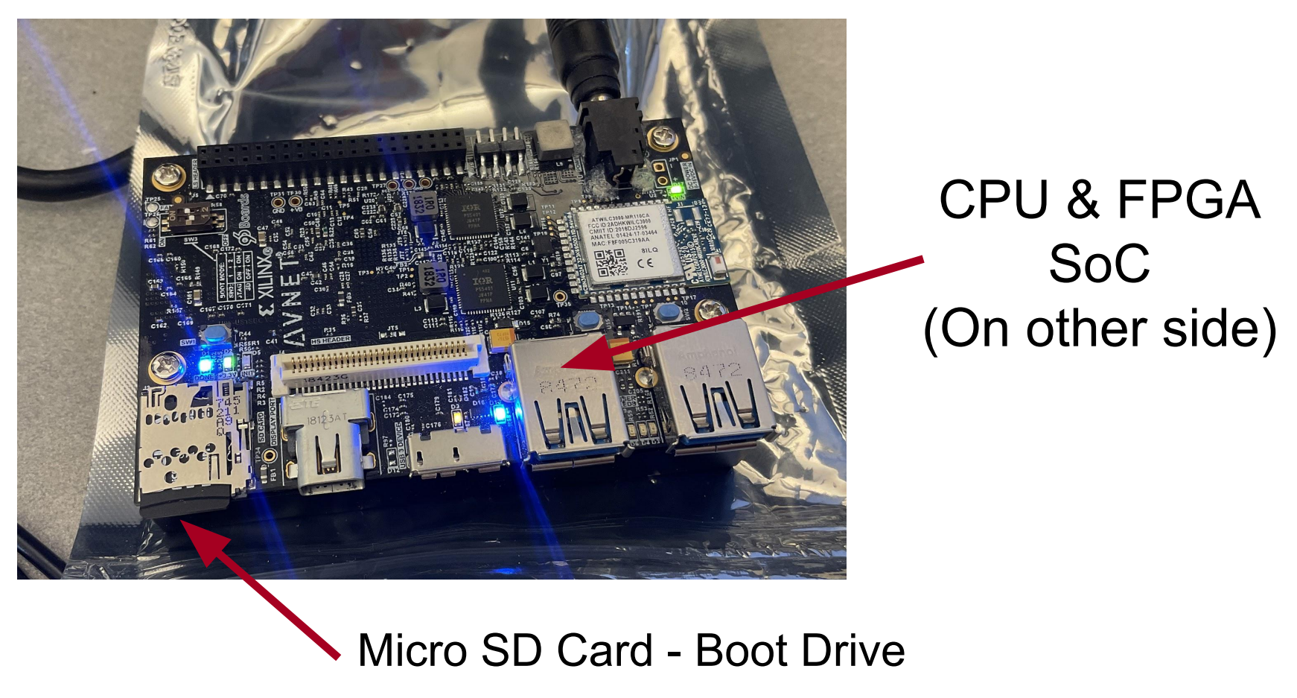
http://course.ece.cmu.edu/~ece500/projects/s22-teame1/

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. It would be nicer in the future to create a more polished interface and allow for a larger set of inputs to be processed.

Our project combined the disciplines of software systems and signal processing for the actual implementation of the fluid simulation algorithm, and hardware systems for the FPGA optimizations. We were able to separately work on individual optimizations and porting the algorithm to the FPGA, but needed to work together for how different portions of the algorithm communicate together and general design strategies.

Electrical & Computer ENGINEERING
Carnegie Mellon

## System Description



Our system supports up to 512 particles per simulation and supports a density of 8194 voxels cubed. The number of particles is a standard size for Scotty3D's simulation size, and the voxel count was chosen based on the limited hardware resources of the Ultra96.

In our system users prepare inputs off-board and are able to upload them via SCP. Then, the accelerated fluid simulation kernel is able to run on these inputs and produces an output file that can be viewed/post-processed by the user offline.

## System Evaluation (Accuracy)

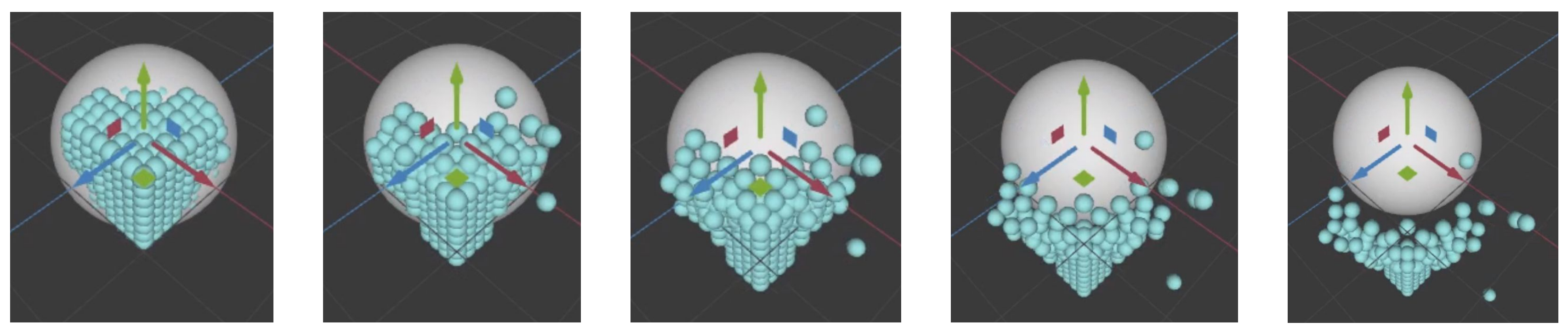
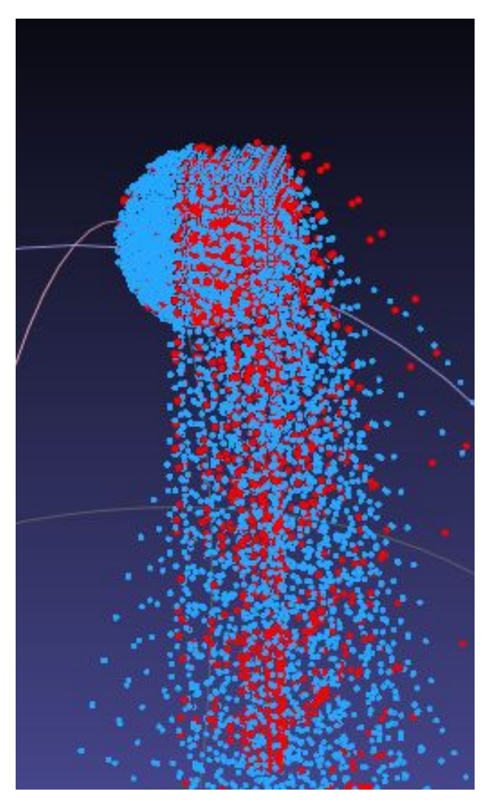


Figure 2: Animation Output from FPGA system



We used the Chamfer Distance metric to compare the output of the FPGA simulation with the CPU simulation (treating fluid simulation traces as point clouds). We compared our Chamfer distances to a reference fluid simulation paper's via ratio of Chamfer distance to particle diameter. We found that our optimized fluid simulation output is sufficiently accurate.

Figure 3 (left): Visualization of Fluid Simulations as Point Clouds

| | Chamfer Distance (m) | Particle Diameter (m) | Ratio (CD/PD) |
|---|---|---|---|
| Ours | 0.26 | 0.125 | 0.48 |
| Reference | 0.03 | 0.05 | 0.6 |

Table 2: Algorithm Accuracy for Benchmark Scene

## System Evaluation (Performance)

Observing the table below, as we apply the optimizations described in the System Architecture section, we see a decrease in the worst case runtime of our kernel. From our results, we determined that enabling pipelining was the most effective optimization we made. Still, we can see that even without any micro-optimizations, we can still achieve a 25x speedup over the original CPU implementation. Still, the spectre of Amdahl's Law still looms over us, as our optimizations were eventually bottlenecked by data transfer bandwidth limitations.

| Changes | Latency (Cycles) | Cycle Time (ns) | **Kernel Runtime (ms)*** | **Timed (ms)** |
|---|---|---|---|---|
| Baseline CPU | - | - | **-** | **15000** |
| Baseline | 4,552,408,073 | 10 | **4,552.41** | **603**** |
| Unrolling | 4,550,196,233 | 10 | **4,550.20** | **602.612** |
| Pipelining | 264,654,053 | 10 | **264.65** | **273.729** |
| Both | 200,889,533 | 10 | **200.89** | **250**** |

Table 3: Performance increases from optimizations
** As reported by Vitis HLS when estimating variable loop counts*
*** Estimated*

Furthermore, as we apply more of our speedups, 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 lower and lower runtimes, we use up more of these resources. 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.

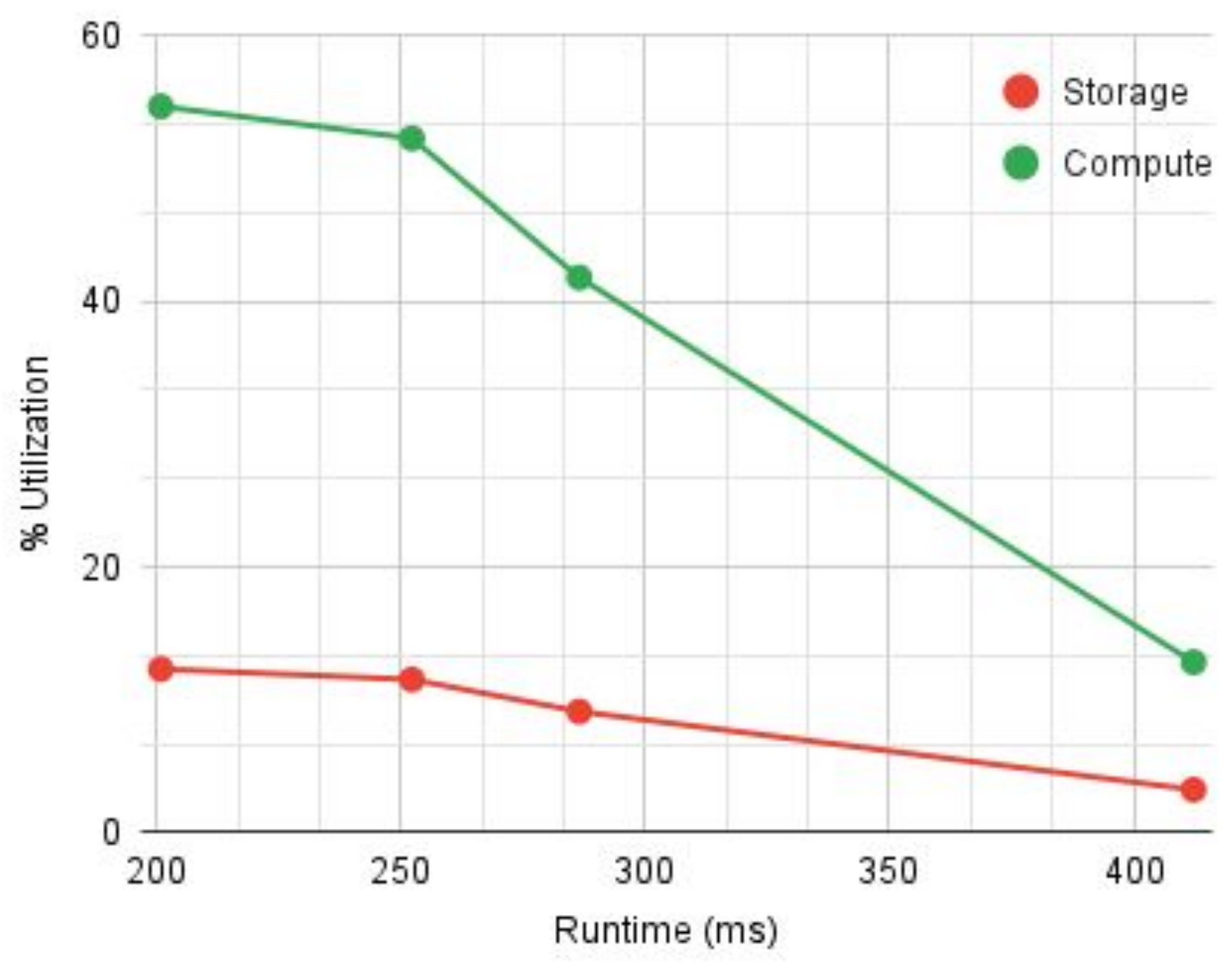


Chart 1: Runtime vs. Resource Utilization