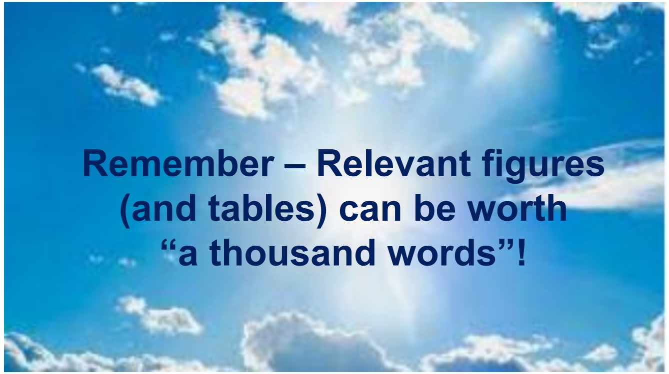
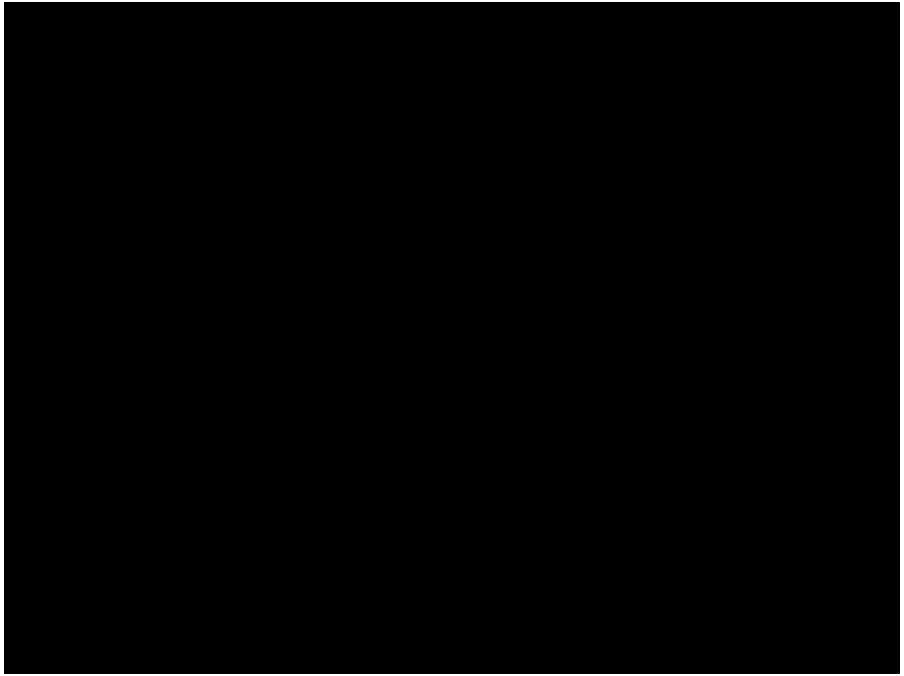
Team E1 - FPGA Accelerated Fluid Simulation

Jeremy Dropkin, Alice Lai, Ziyi Zuo

Add your 12 slides after this slide… [remember, 12 min talk + 3 min Q/A]



Use Case



- Simulating fluids on the CPU is slow due to large number of computations and large number of particles

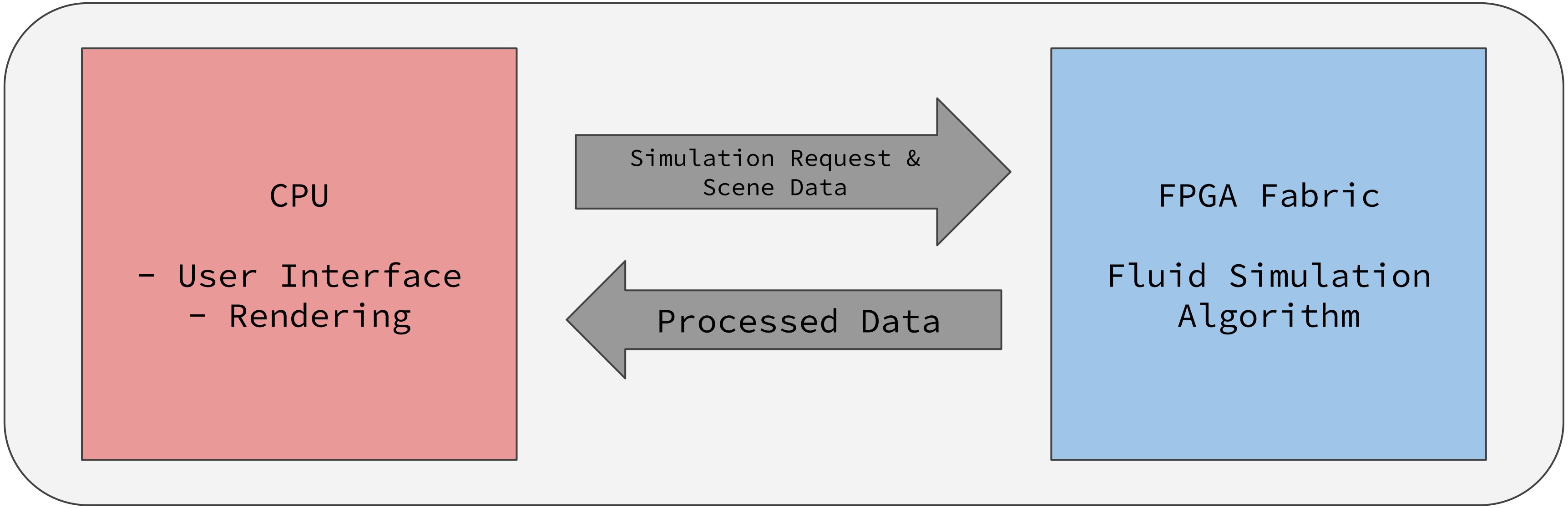
- We want to simulate fluids on the FPGA to take advantage of the FPGA’s parallelism and provide significant speedup with low power consumption

Use Case - Quantitative Requirements

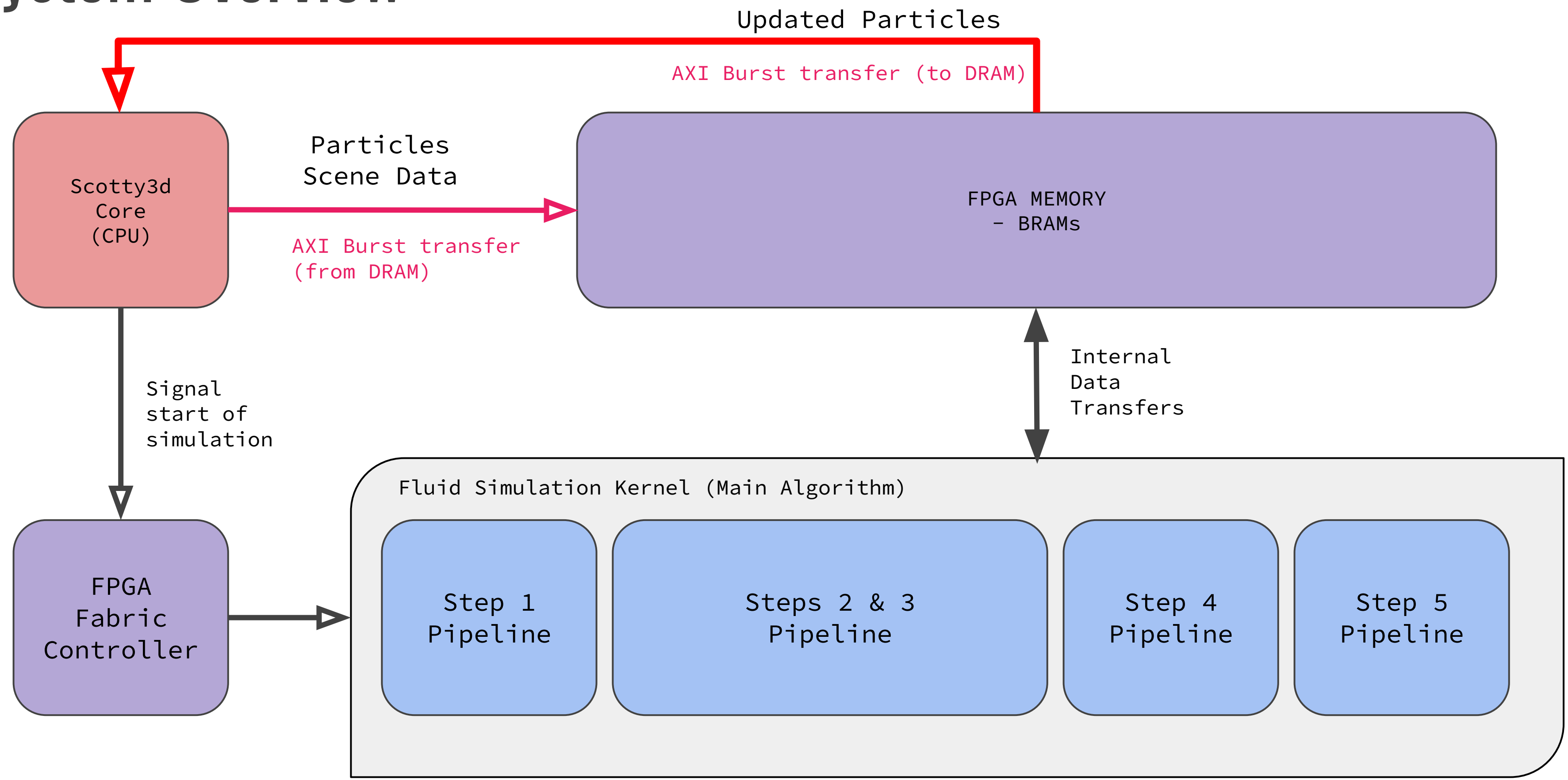
- Baseline
  - ~3 second render time (per frame) on an i7-8665U @ 1.90 GHz x 8
- Our goal is to make at least a **10x speedup** for simulating a fluid of size **512 particles**
  - Speedup Motivation:
    - Ultra96v2 Fabric Clock ~150MHz, ~13x slower than the i7-8665U
    - Much of the compute task is data movement
      - Cache/DRAM -> 10s/1000s of cycles; SRAM -> 1s of cycles
    - Multiplying these factors together gets ~10x parallelism/speedup
  - Number of Particles Motivation:
    - 512 particles is the standard size for Scotty3D fluid simulations

Solution Approach

- Accelerate Fluid Simulation Algorithm on the FPGA fabric
- CPU handles the rest of the Scotty3D rendering stack



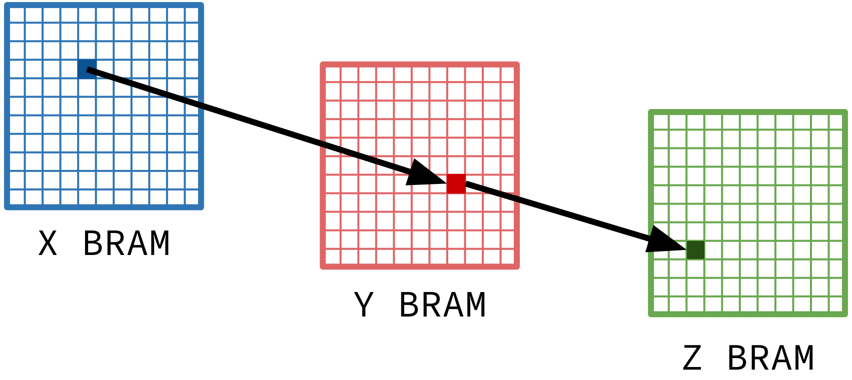
System Overview



Fluid Simulation Kernel Optimizations

Step 1: 3-Level Hardware Map

- 3D Point Lookup
- Need an efficient implementation of hashmap for BRAM



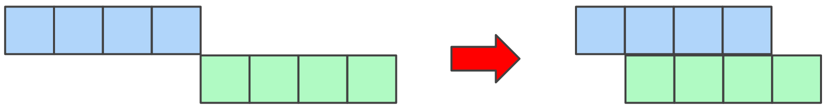
Algorithm 1 Simulation Loop

for all particles i **do**
    apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
    predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for
for all particles i **do**
    find neighboring particles $N_i(\mathbf{x}_i^*)$ — Step 1
end for
while $iter < solverIterations$ **do**
    for all particles i **do**
        calculate λ_i
    end for
    for all particles i **do**
        calculate $\Delta \mathbf{p}_i$
        perform collision detection and response
    end for
    for all particles i **do**
        update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$
    end for
end while
for all particles i **do**
    update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
    apply vorticity confinement and XSPH viscosity
    update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for

Fluid Simulation Kernel Optimizations

Steps 2 & 3: Pipelining

- We can pipeline different chunks together if they are independent of each other!



Algorithm 1 Simulation Loop

for all particles i **do**
  apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
  predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for

Step 1
for all particles i **do**
  find neighboring particles $N_i(\mathbf{x}_i^*)$
end for

while $iter < solverIterations$ **do**

Step 2
  for all particles i **do**
    calculate λ_i
  end for

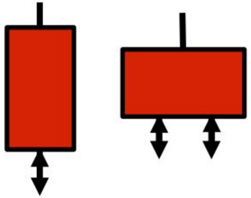
Step 3
  for all particles i **do**
    calculate $\Delta \mathbf{p}_i$
    perform collision detection and response
  end for

  for all particles i **do**
    update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$
  end for
end while
for all particles i **do**
  update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
  apply vorticity confinement and XSPH viscosity
  update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for

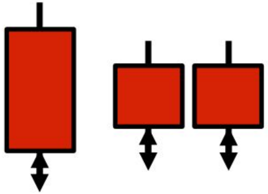
Fluid Simulation Kernel Optimizations

Steps 4 & 5: Use Block RAM (BRAM)

- We want to avoid contention of the memory devices where the particles are stored
  - Create copies of data
    - Increase accessibility
  - Modify BRAM arrays
    - Reshape - Widen ports



    - Partition - Split banks

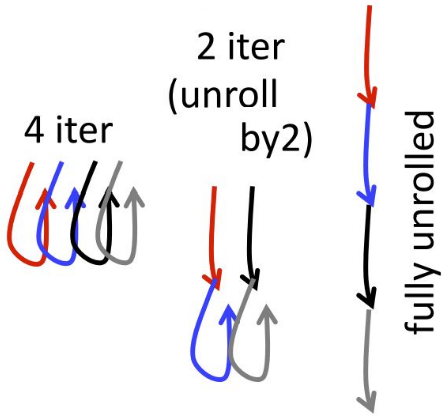


Algorithm 1 Simulation Loop

for all particles i **do**
  apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
  predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for
for all particles i **do** — Step 1
  find neighboring particles $N_i(\mathbf{x}_i^*)$
end for
while $iter < solverIterations$ **do**
  for all particles i **do** — Step 2
    calculate λ_i
  end for
  for all particles i **do** — Step 3
    calculate $\Delta\mathbf{p}_i$
    perform collision detection and response
  end for
  for all particles i **do** — Step 4
    update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta\mathbf{p}_i$
  end for
end while
for all particles i **do** — Step 5
  update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
  apply vorticity confinement and XSPH viscosity
  update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for

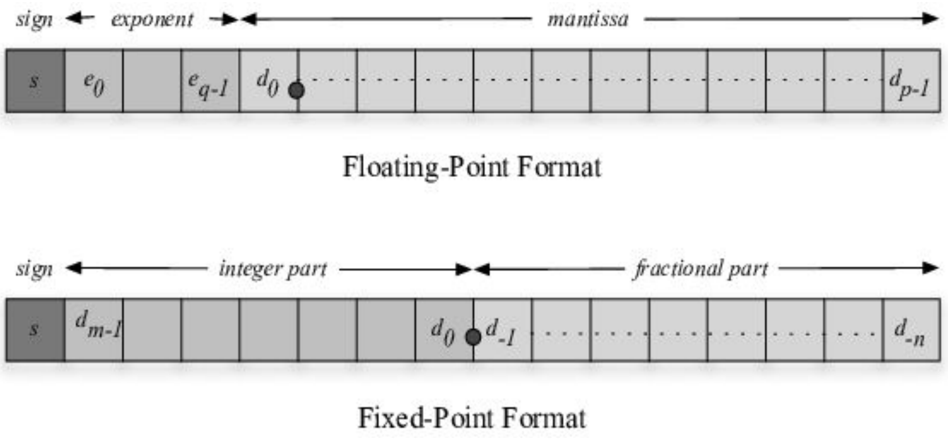
General Optimizations

- Unrolling
  - Instantiate more hardware to increase concurrency
  - Run each iteration in parallel



General Optimizations

- Fixed point numbers instead of floating point
  - Floating point numbers requires lining up the floating point
  - Fixed point numbers are stored as ints - faster and more optimal

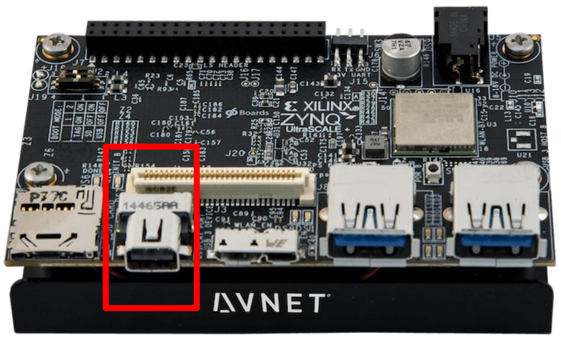


Implementation Plan

- Xilinx Ultra96 FPGA Platform
  - Vitis High Level Synthesis (HLS) → Generate hardware from C/C++ code

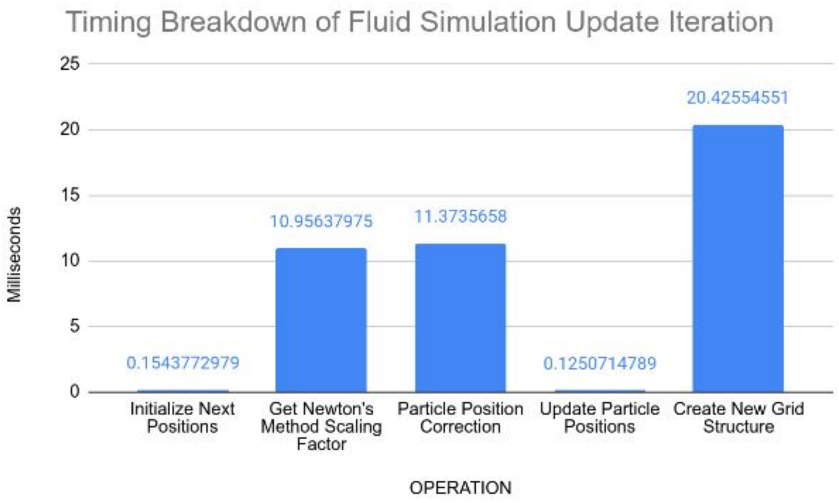
- Scotty3D codebase

- Mini Display Port cable for Visualization (Purchase)



Testing & Verification

- Quantitative evaluation:
  - Reduce the aggregate runtime of steps 2 & 3 by 50%
  - Maintain pace with runtime of steps 1 & 4 (relative to baseline)
  - Reduce runtime of step 5 by 20%
  - Comparison of resulting data against a golden result
    - Goal: **90%** accuracy



- Qualitative evaluation:
  - Visual inspection to ensure rendered animations still retain “fluid-like” quality

Schedule & Division of Labor