

Name: Ty Torres

UNI: tt3019

pic-flip

I propose for this assignment to parallelize a PIC/FLIP fluid method in haskell. Since there are no repositories I found that have already implemented it I will code a single-threaded version with AI-assistance. My plan for the project is to firstly get the single threaded version running since it's a large task already with the particles, grids, and boundary handling. Then to parallelize all the movement of information and calculating of simple and complex information as discussed further in the next paragraph.

The algorithm for PIC/FLIP uses a combination of the PIC, particle in cell method, and the FLIP, fluid implicit particle methods to update the velocity. It utilizes the Navier Stokes equation with a hybrid method using Lagrangian particles and an Eulerian grid. It works by using particles in cells and a staggered grid that stores velocities and pressures transferred from the particles. There are 3 grids in total one for the horizontal velocities, one for the vertical velocities and lastly one for the pressure of each current cell. Viscosity can be excluded since we are explicitly doing a water simulation. So it then simplifies to the following:

$$\partial \mathbf{u} / \partial t + (\mathbf{u} \cdot \nabla) \mathbf{u} = -(1/\rho) \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

$$\partial \mathbf{u} / \partial t + (\mathbf{u} \cdot \nabla) \mathbf{u} = -(1/\rho) \nabla p + \mathbf{f}$$

The main loop then proceeds as follows:

1. Add forces to each particle and step through using symplectic euler to the next position correcting any collisions. (advection portion)
2. Transfer particle velocities to the grid using weighted averages, separately for horizontal and vertical components. Save a copy of this grid.
3. Mark cells as fluid, air, or solid

4. Solve the system of equations for the pressure in each fluid cell, using the divergence of the grid velocities, to make the water incompressible.

- a. $\mathbf{0} = (\mathbf{u} \cdot \nabla)$ (incompressibility constraint)

- b. $\nabla^2 \mathbf{p} = -\rho \nabla \cdot \mathbf{u}_{\text{new}}$ (equation used to solve for p)

5. Use the new pressure values to correct the velocities at each point by using the difference between their neighbors.
6. Update particle velocities with $0.9 \times \text{FLIP} + 0.1 \times \text{PIC}$, where PIC bilinearly interpolates the new grid and FLIP adds the interpolated change (new – saved) to the particle’s current velocity.

Parallelization plan. After building and verifying a serial baseline, I will parallelize the stages by how much they depend on each other:

- Independent per-particle stages (applying forces, advecting particles, grid-to-particle transfer): parallel maps over chunks of particles using Strategies.
- Pressure solve: the most expensive stage, since each cell depends on its neighbors over many iterations. I will compare a Jacobi solver, parallelized per cell with Repa, against red-black Gauss-Seidel, which converges faster but must alternate between two half-grids.
- Particle-to-grid transfer: many particles write to the same grid faces, causing write conflicts. I will use the Par monad to give each task a private grid and sum the grids afterward, and if time allows, compare this against binning particles by cell so each face gathers its own contributions with no conflicts.

I will measure speedup on 1, 2, 4, and 8 cores, both per stage and overall, and use ThreadScope to find bottlenecks.

Input Data: Not needed since it will provide its own initial conditions.

Scope: This will be in 2D and If time runs short, I will keep the simpler versions (per-task grids for particle-to-grid, Jacobi for pressure) instead of the comparisons.