Optimization of a FLIP algorithm

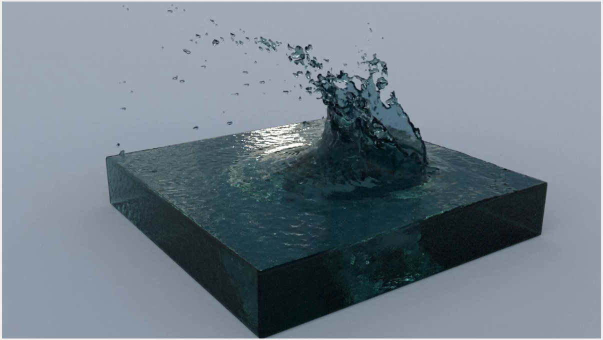
ETHZ – Advanced System Lab Project

Chris Amevor
Sean Bone
Felix Illes
Mikael Stellio

June 7, 2021

The FLIP algorithm – Introduction

- FLIP: **Fl**uid **I**mplicit **P**article
- Hybrid particle- and grid-based method
- Commonly used in computer graphics to simulate fluids



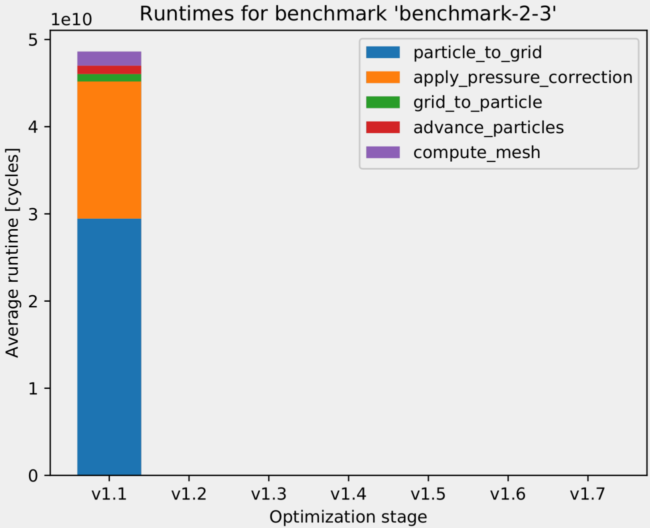
The FLIP algorithm – Components

At each timestep it

1. Computes the velocity field by particle-to-grid projection,
2. Applies external forces to the velocity field,
3. Enforces boundary conditions,
4. Computes pressure gradients and updates the velocity field,
5. Updates particle velocities using grid-to-particle projection,
6. Advects particles using 2nd-order Runge-Kutta integration,
7. Computes a level set & mesh around particles.

Optimizations

Initial Runtime

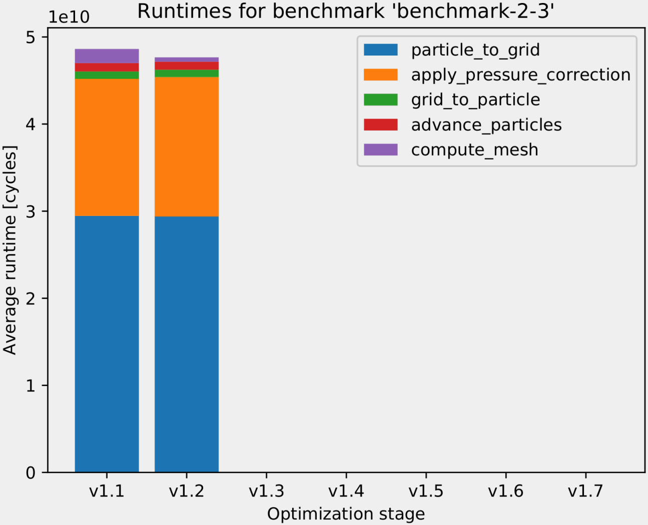


Optimizations – Level Set Computation

The major benefits came from:

- Skipping construction of Eigen vectors: work directly on arrays.
- Providing `Eigen::Map` objects where needed (Marching Cubes implementation).
- Computing boundary cells separately, simplifying inner loop.
- Inlining methods.
- Arithmetic optimizations & strength reduction: skip square root by working with $|x|^2$, constant div $\rightarrow$ mul.

Runtime – Level Set



- Less bandwidth wasted
- Improved locality
- No "compiler black box" effect

Array of structs

↓

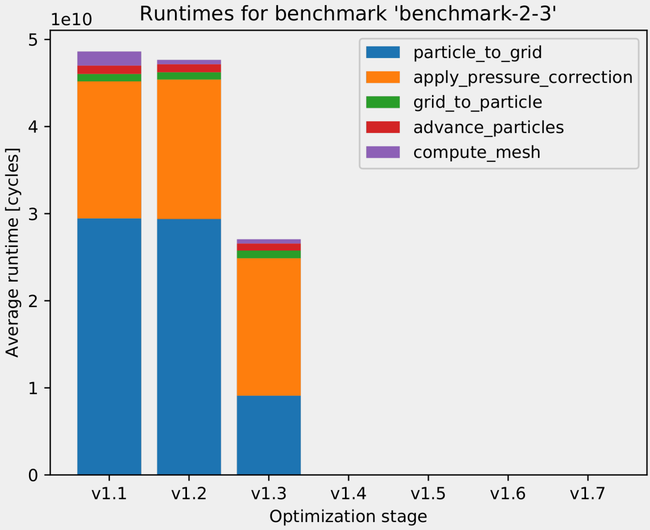
Struct of arrays

Optimizations – Particles data structure

Two more optimizations were attempted without success (so far):

- **Particles sorting:** improves locality, but causes overhead!
- **Cell index caching:** fewer flops at cost of more memory.

Runtime – Particles data structure



Optimizations – Advection and grid-to-particle

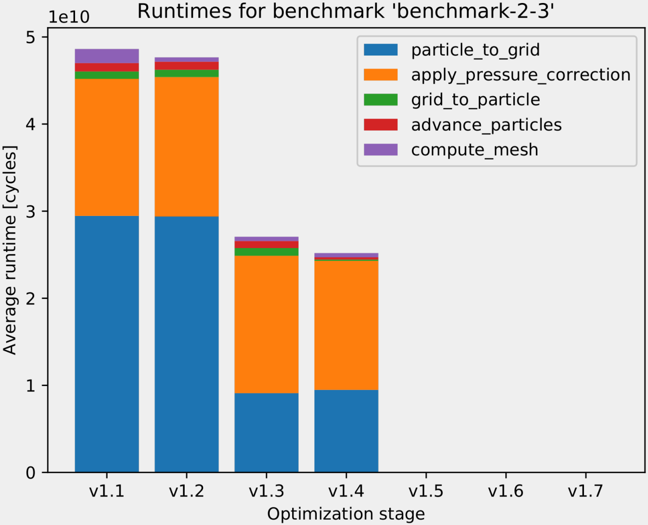
The main performance gain for both of these was in rewriting the velocity interpolation routine:

- Fewer function calls.
- Simpler logic and less branching.
- Template expressions to "generate" different versions for U, V, and W.
- Fused U and U* velocities interpolation in a single call. This again was done with templates and benefits grid-to-particle.
- Strength reduction: dividing by cell size becomes multiplication.

Manual vectorization is not beneficial:

- Vectorizing just the trilinear interpolation kernel manually is a performance loss.
- Vectorizing the whole interpolation routine is infeasible due to complex branching (18 branches!).
- Vectorizing advection or grid-to-particle methods might be possible; however, they are dominated by interpolation.
- Because of the low footprint of advection and grid-to-particle, further optimization is deemed unnecessary.

Runtime – Advection and grid-to-particle



The major benefits came from:

- Manually inlining functions to allow loop fusion,
- Separating the accumulation loop for boundary cells,
- Minimizing conditionals inside loops.

Velocity fields U, V, and W have different dimensions!

↓

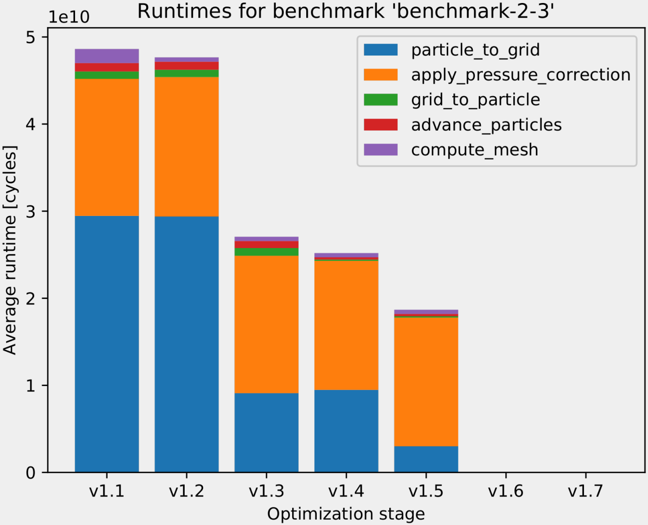
Large overhead to avoid out-of-range accesses
(exploit ghost cells?)

Not very prone to vectorization and unrolling, due to:

- Unordered particles on the grid
- Unknown number of particles in a grid-cell

However, the normalization of accumulated velocities was completely vectorized resulting in a small speedup!

Runtime – Particle-to-grid



Optimizations – Pressure correction

Original implementation using sparse Eigen solver, requiring construction of sparse s.p.s.d. matrix A, where

$$A_{i,j} = A_{j,i} = \begin{cases} n_i & i = j, \\ -1 & \text{cells i and j adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

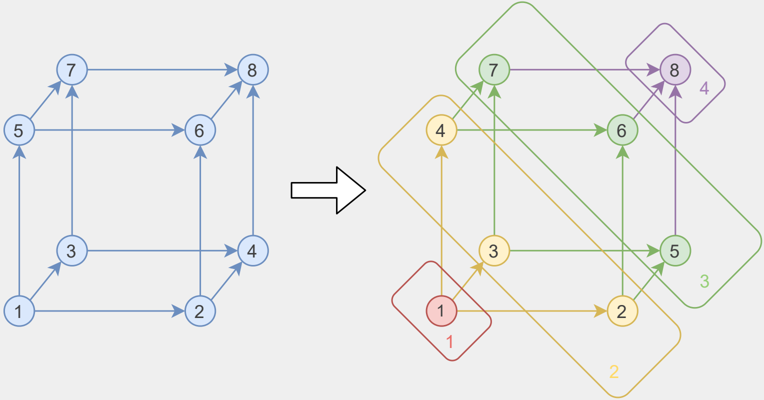
Optimizations – Pressure correction

A custom ICCG solver allowed for better performance:

- No explicit representation of A, significantly decreased memory usage & bandwidth.
- Fused loops where possible.
- Vectorization of kernels.

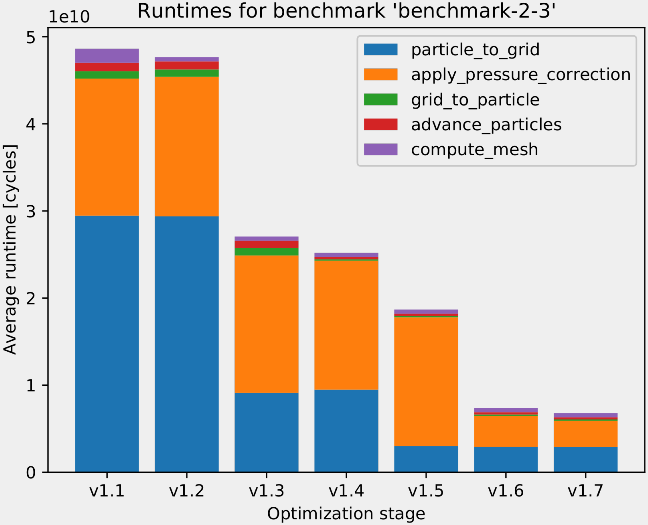
Optimizations – Pressure correction

Blocking of forward & backward substitution to increase ILP was not beneficial.



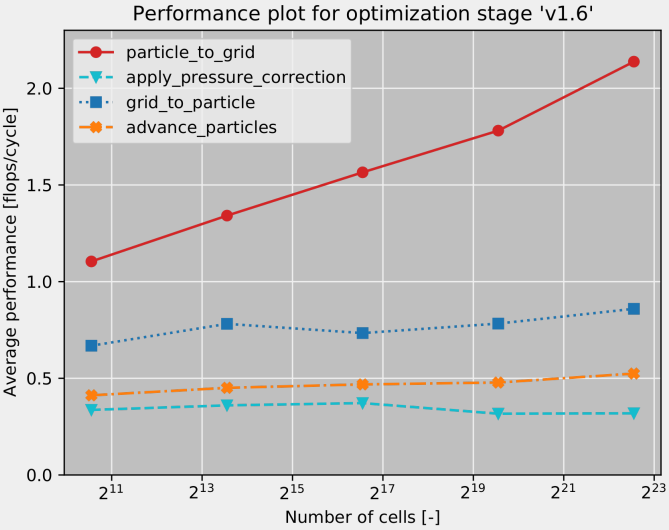
Problem: increases possible ILP at the cost of higher memory bandwidth!

Runtime – Pressure correction

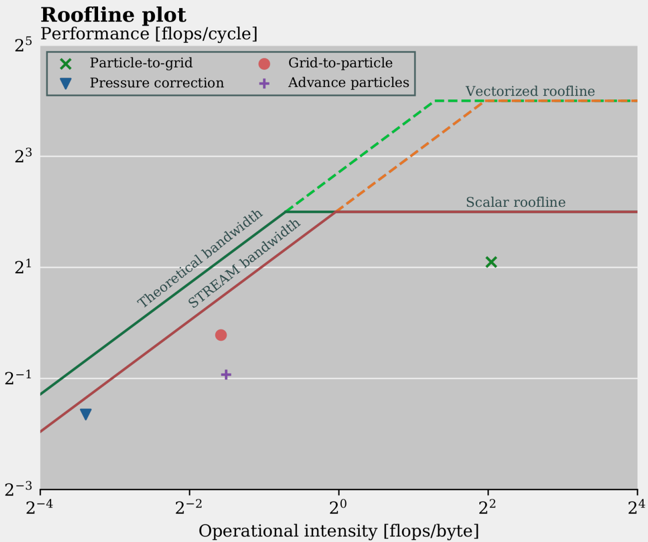


Results

Results – Performance



Results – Roofline



Thanks for your attention!