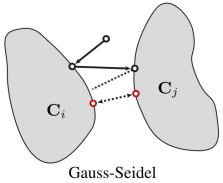
Position Based Dynamics

Team 32
Thomas Creavin, Clément Jambon, Manuel De Prada Corral, Marius Debussche

Algorithm & Implementation

Reminder on PBD and our implementation



Gauss-Seidel

Algorithm 1 Position Based Dynamics

1: INITIALIZE($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
2: **loop**
3: APPLYEXTERNALFORCES($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
4: INTEGRATEVELOCITIES($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
5: GENERATECOLLISIONCONSTRAINTS($\mathbf{p}_1, \ldots, \mathbf{p}_N$)
6: **for** solverIterations **do**
7: PROJECTCONSTRAINTS($\mathbf{p}_1, \ldots, \mathbf{p}_N$)
8: **end for**
9: UPDATESTATE($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
10: **end loop**

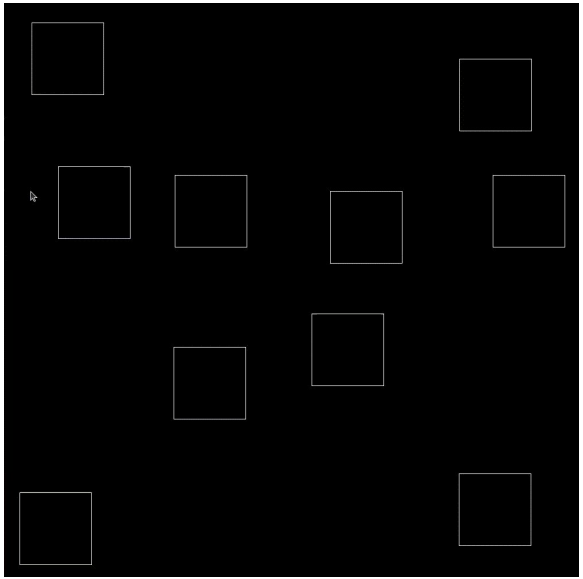
Input: a number n of quads (initialized as rectangles) each with two side-lengths, an initial position and rotation for each rectangle, a simulation duration *T* and a step size *dt* (and additional simulation hyperparameters)

Output: the trajectory of each vertex of the quads subject to 2 types of constraints: constitutive constraints (per-quad *stretching* & *shearing*) and collision constraints (*vertex-to-edge* & *edge-to-edge*)

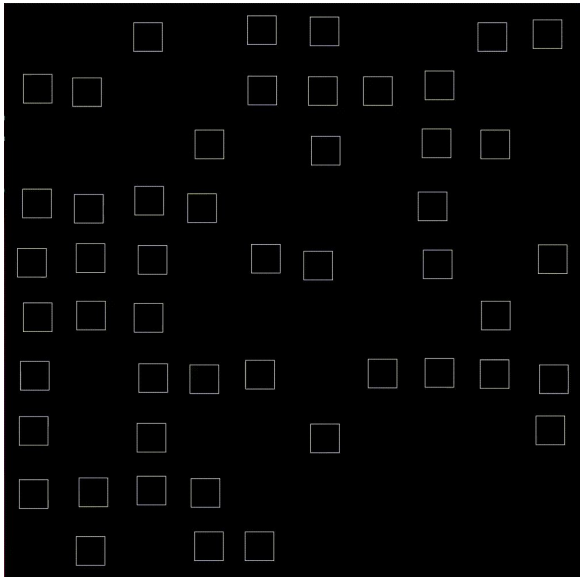
Main takeaways:

- PBD targets efficiency and plausibility
- The "solver" is a purely iterative algorithm (not a linear solver!)
- constraint projections are O(n) while detection is $O(n^2)$
- every update is performed sequentially (cf. Gauss-Seidel analogy)
- although deterministic w.r.t. the initial conditions, simulations are highly heterogeneous (e.g., collisions are sparsely satisfied)

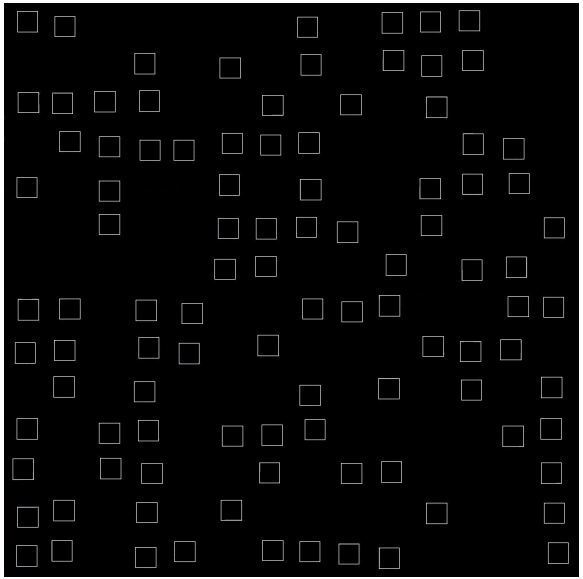
Validation



n=10



n=50



n=100

Analysis

Cost analysis

```
void update_collision_constraint(Vector2D *q, Vector2D *p1, Vector2D *p2, fp w) {
    q->x += ((p1->y - p2->y) *
             (p1->y * (p2->x - q->x) + p2->y * q->x - p2->x * q->y + p1->x * (-p2->y + q->y)) *
             POW(POW(ABS(p1->x - p2->x), 2) + POW(ABS(p1->y - p2->y), 2), 2)) /
            (POW(ABS(p1->x - p2->x), 6) +
             3 * POW(ABS(p1->x - p2->x), 4) * POW(ABS(p1->y - p2->y), 2) +
             3 * POW(ABS(p1->x - p2->x), 2) * POW(ABS(p1->y - p2->y), 4) +
             POW(ABS(p1->y - p2->y), 6) +
             POW(ABS((-p1->y + q->y) * POW(ABS(p1->x - p2->x), 2) +
                     (-p1->y + q->y) * POW(ABS(p1->y - p2->y), 2) +
                     (p1->y * p2->x - p1->x * p2->y - p1->y * q->x + p2->y * q->x + p1->x * q->y -
                      p2->x * q->y) *
                       ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
                 2) +
             POW(ABS((p2->y - q->y) * POW(ABS(p1->x - p2->x), 2) +
                     (p2->y - q->y) * POW(ABS(p1->y - p2->y), 2) +
                     (-(p1->y * p2->x) + p1->x * p2->y + p1->y * q->x - p2->y * q->x -
                      p1->x * q->y + p2->x * q->y) *
                       ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
                 2) +
             POW(ABS((-p2->x + q->x) * POW(ABS(p1->x - p2->x), 2) +
                     ABS(p1->y - p2->y) * ((-p2->x + q->x) * ABS(p1->y - p2->y) +
                                           (p1->y * p2->x - p1->x * p2->y - p1->y * q->x +
                                            p2->y * q->x + p1->x * q->y - p2->x * q->y) *
                                             sign(p1->y - p2->y))),
                 2) +
             POW(ABS((p1->x - q->x) * POW(ABS(p1->x - p2->x), 2) +
                     ABS(p1->y - p2->y) * ((p1->x - q->x) * ABS(p1->y - p2->y) +
                                           (-(p1->y * p2->x) + p1->x * p2->y + p1->y * q->x -
                                            p2->y * q->x - p1->x * q->y + p2->x * q->y) *
                                             sign(p1->y - p2->y))),
                 2));

    /* REPEATED 5 MORE TIMES */
}
```

```
void update_midpoint_constraint(Vector2D *q1, Vector2D *q2, Vector2D *p1, Vector2D *p2, fp w) {
    q1->x += -(
        ((p1->y - p2->y) *
         (-(p2->y * q1->x) + p2->x * q1->y - p2->y * q2->x + p1->y * (-2 * p2->x + q1->x + q2->x) +
          p1->x * (2 * p2->y - q1->y - q2->y) + p2->x * q2->y) *
         POW(POW(ABS(p1->x - p2->x), 2) + POW(ABS(p1->y - p2->y), 2), 2)) /
        (2 * POW(ABS(p1->x - p2->x), 6) +
         6 * POW(ABS(p1->x - p2->x), 4) * POW(ABS(p1->y - p2->y), 2) +
         6 * POW(ABS(p1->x - p2->x), 2) * POW(ABS(p1->y - p2->y), 4) +
         2 * POW(ABS(p1->y - p2->y), 6) +
         POW(ABS((2 * p2->y - q1->y - q2->y) * POW(ABS(p1->x - p2->x), 2) +
                 (2 * p2->y - q1->y - q2->y) * POW(ABS(p1->y - p2->y), 2) +
                 (-(p2->y * q1->x) + p2->x * q1->y - p2->y * q2->x +
                  p1->y * (-2 * p2->x + q1->x + q2->x) + p1->x * (2 * p2->y - q1->y - q2->y) +
                  p2->x * q2->y) *
                    ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
             2) +
         POW(ABS((-2 * p1->y + q1->y + q2->y) * POW(ABS(p1->x - p2->x), 2) +
                 (-2 * p1->y + q1->y + q2->y) * POW(ABS(p1->y - p2->y), 2) +
                 (p2->y * q1->x - p2->x * q1->y + p1->y * (2 * p2->x - q1->x - q2->x) +
                  p2->y * q2->x - p2->x * q2->y + p1->x * (-2 * p2->y + q1->y + q2->y)) *
                    ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
             2) +
         POW(ABS((2 * p1->x - q1->x - q2->x) * POW(ABS(p1->x - p2->x), 2) +
                 ABS(p1->y - p2->y) * ((2 * p1->x - q1->x - q2->x) * ABS(p1->y - p2->y) +
                                       (-(p2->y * q1->x) + p2->x * q1->y - p2->y * q2->x +
                                        p1->y * (-2 * p2->x + q1->x + q2->x) +
                                        p1->x * (2 * p2->y - q1->y - q2->y) + p2->x * q2->y) *
                                         sign(p1->y - p2->y))),
             2) +
         POW(ABS((-2 * p2->x + q1->x + q2->x) * POW(ABS(p1->x - p2->x), 2) +
                 ABS(p1->y - p2->y) *
                   ((-2 * p2->x + q1->x + q2->x) * ABS(p1->y - p2->y) +
                    (p2->y * q1->x - p2->x * q1->y + p1->y * (2 * p2->x - q1->x - q2->x) +
                     p2->y * q2->x - p2->x * q2->y + p1->x * (-2 * p2->y + q1->y + q2->y)) *
                       sign(p1->y - p2->y))),
             2)));

    /* REPEATED 7 MORE TIMES */
}
```

Cost analysis

After a **manual** count of each function:

Function	Flops	Function	Flops
add_toy_velocity	4n	ray_segment_intersection	~14
apply_external_forces	8n	shearing	60
damp_velocities	97n	stretching	120
init_simulation	30n + 8	toy_constraint	16
inside_quad_test	60	update_collision_constraint	1065
integrate_velocities	16n	update_midpoint_constraint	1912
ray_quad_intersection	~61	update_state	16n

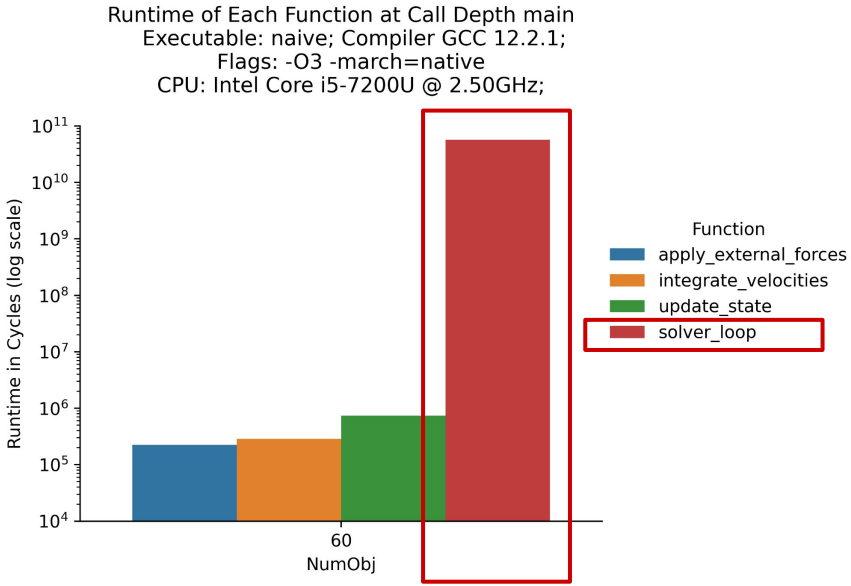
Called $O(n^2)$

Called $O(n^2)$ if generating constraints every solver loop ($O(n)$ otherwise)

Two granularities of analysis

1. **Macro-benchmarks** on real simulation scenarios
   - **Auto-generated** simulation scenarios (for each input size)
   - As constraints (e.g., collisions) are not always satisfied, we use a fine-grained telemetry to record events in a **first execution pass**; this allows to accurately determine the number of flops. We then run the simulation again a **second to time to record the accurate runtime**.
2. **Micro-benchmarks** to finely identify the benefits of our optimizations
   - **Unit-test** style performance measurement, robust and isolated.
   - Runs on synthetic data.
   - Ensures output **consistency** across all implementations.

Identifying bottlenecks

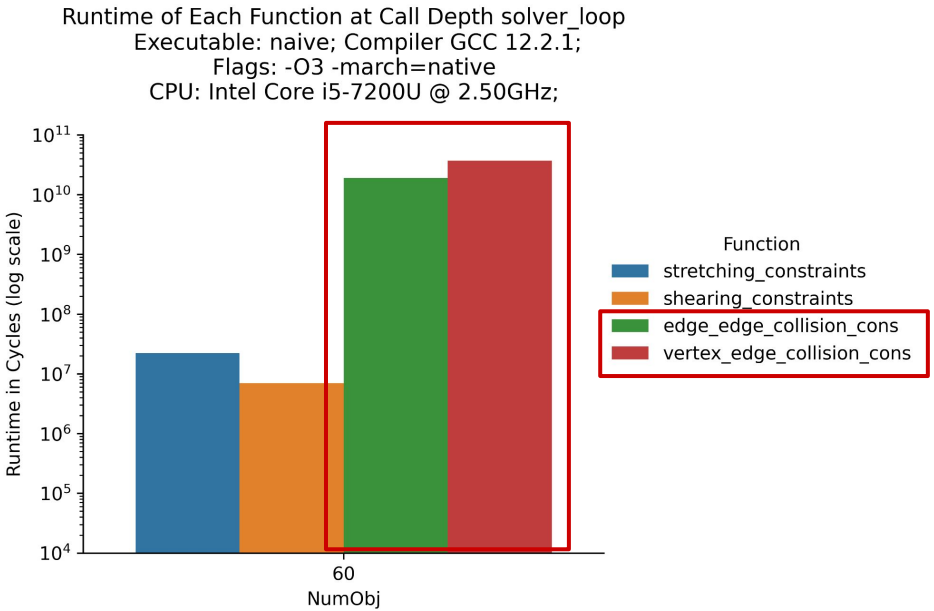


Algorithm 1 Position Based Dynamics

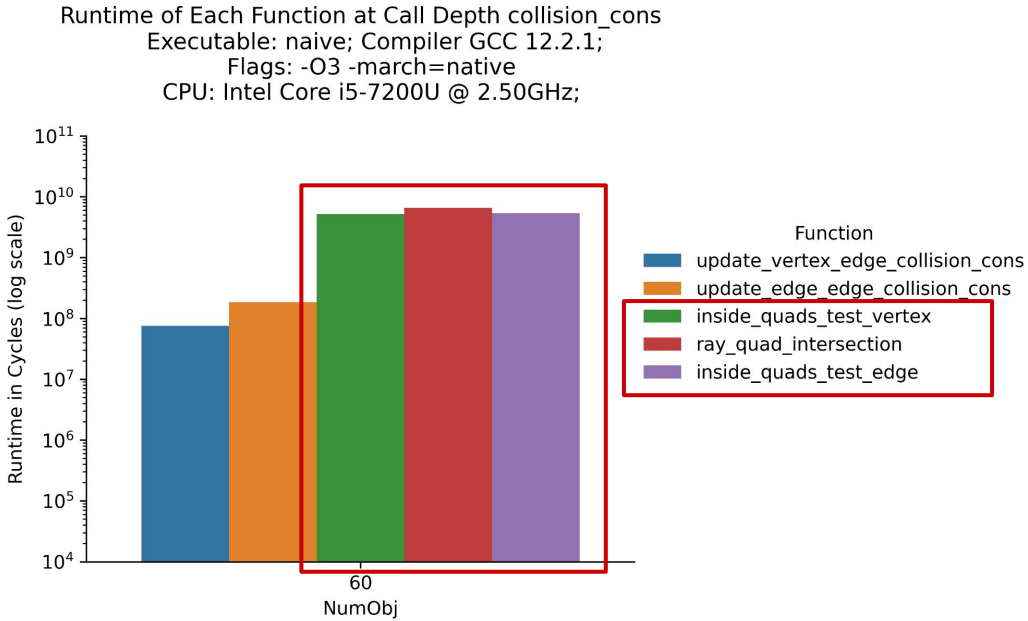
1: INITIALIZE($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
2: **loop**
3: APPLYEXTERNALFORCES($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
4: INTEGRATEVELOCITIES($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
5: GENERATECOLLISIONCONSTRAINTS($\mathbf{p}_1, \ldots, \mathbf{p}_N$)
6: **for** solverIterations **do**
7: PROJECTCONSTRAINTS($\mathbf{p}_1, \ldots, \mathbf{p}_N$)
8: **end for**
9: UPDATESTATE($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathbf{v}_1, \ldots, \mathbf{v}_N$)
10: **end loop**

Identifying bottlenecks

```
1: procedure ProjectConstraints(p_1, ..., p_N)
2:     for i ← 1, N do
3:         StretchingConstraints(p_i)
4:         ShearingConstraints(p_i)
5:         ToyConstraints(p_i)
6:         VertexEdgeCollision(p_i)
7:         EdgeEdgeCollision(p_i)
8:     end for
9: end procedure
```



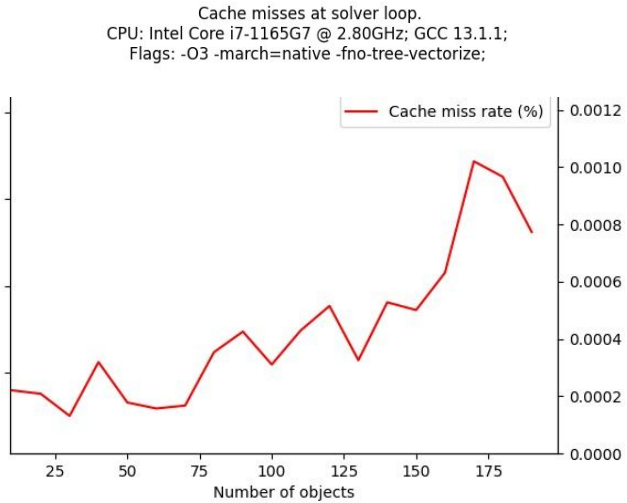
Identifying bottlenecks



$O(n^2)$!!!

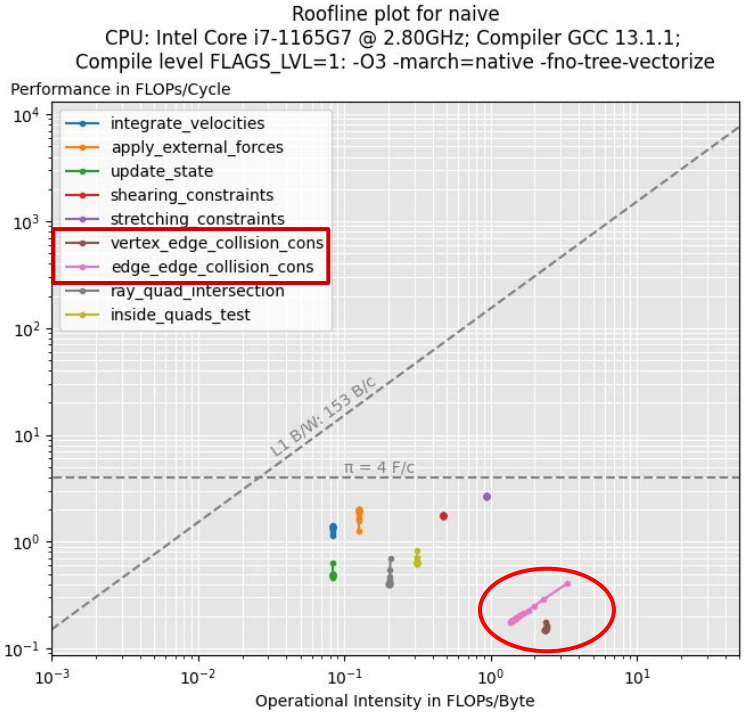
1: **procedure** COLLISIONCONSTRAINT($\mathbf{p}_i$)
2: **for** $j \leftarrow 1, N$ **do**
3: **if** TESTCOLLISION($\mathbf{p}_i$, $\mathbf{p}_j$) **then**
4: RESOLVECOLLISION($\mathbf{p}_i$, $\mathbf{p}_j$)
5: **end if**
6: **end for**
7: **end procedure**

Memory analysis: compute-bound



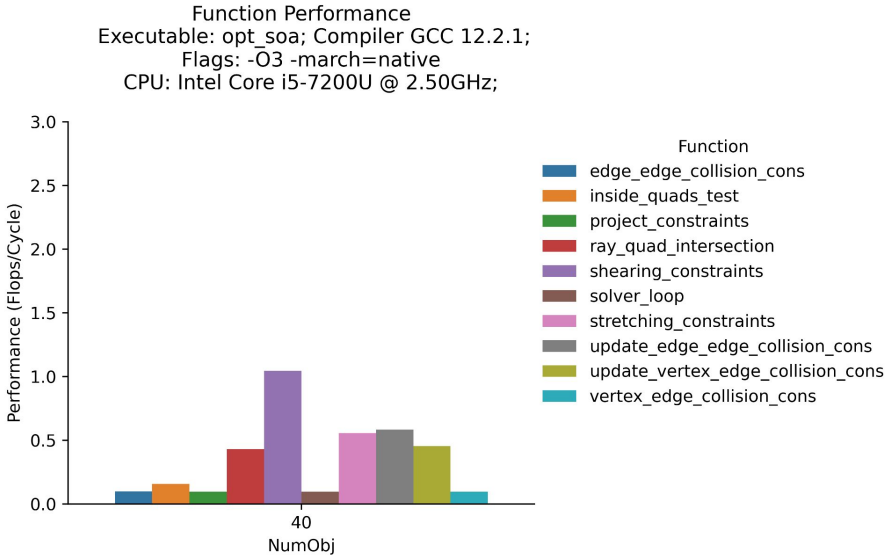
- Memory profiling using Linux Perf Event (LPE) headers.
- Results: miss rate in bottleneck code is close to 0.
- All data fits in L1 cache:
  - 190 quads * 264 bytes per quad = 50KB
  - Experiments on i7 1165G7, L1 cache of 48 KB

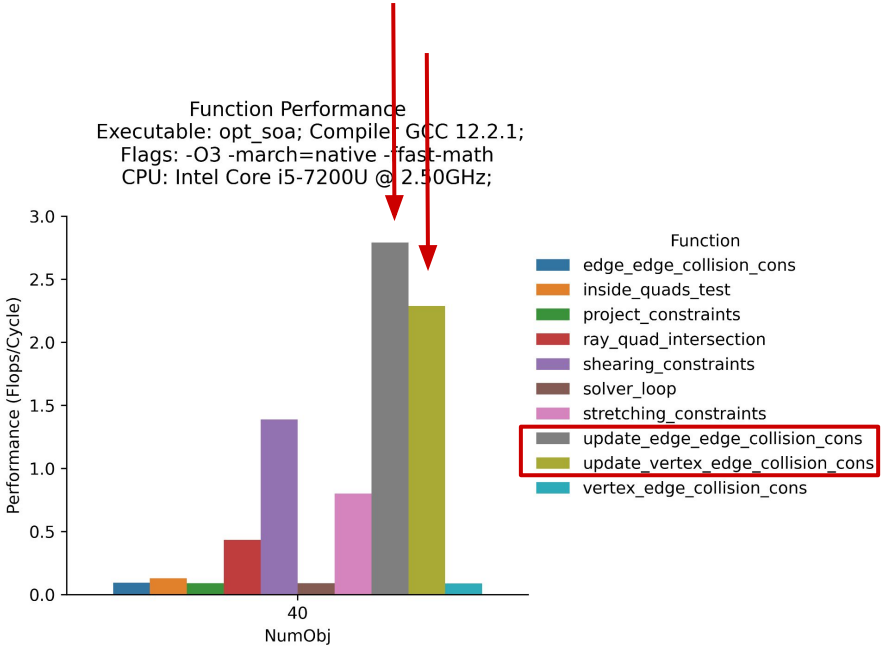
Rooflines



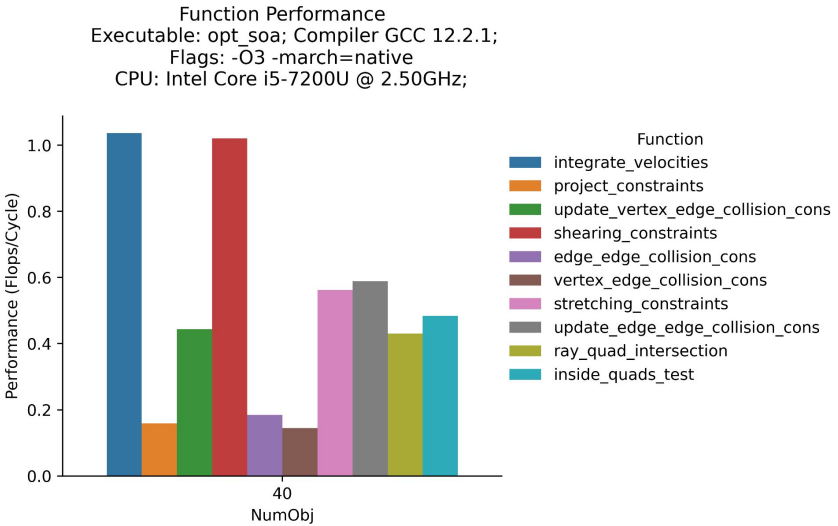
- Roofline plot confirms that we are completely **compute-bound**.
- Bottleneck functions have **high operational intensity**.

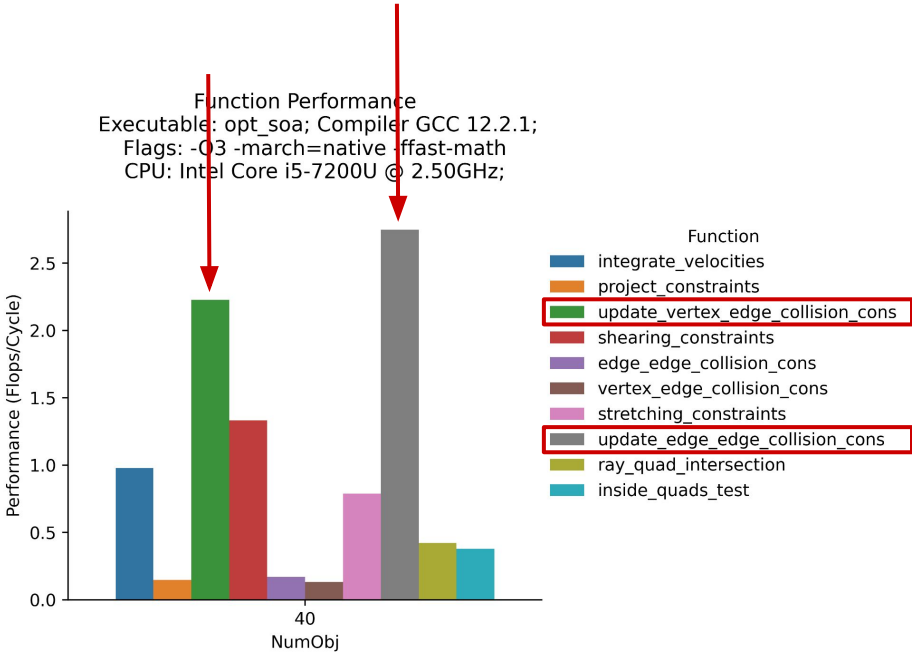
Performance & -ffast-math





Performance & -ffast-math





Optimizations

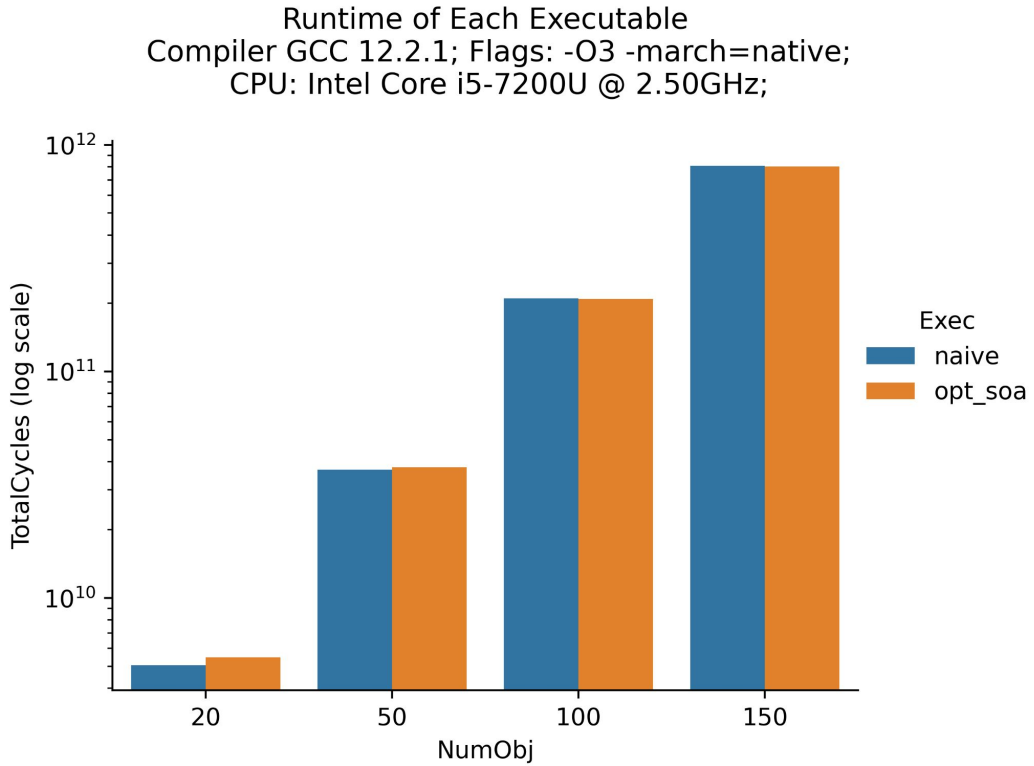
Overview

- Flops reduction
- Precomputation:
  - determine collisions before the projecting constraints
  - precomputing reused values in *generate_collision_constraints*
- Improved data structures
  - AOS to SOA
  - storing precomputed collision records in lightweight buffers
- Vectorization

Failed optimizations:

- optimize the **power functions** (mostly powers of 2, 4 and 6) in collision constraint projections but already heavily optimized in libc
- **scalar replacement** in the collision constraint projections, no significant speed-up

AOS to SOA


Runtime of Each Executable
Compiler GCC 12.2.1; Flags: -O3 -march=native;
CPU: Intel Core i5-7200U @ 2.50GHz;
TotalCycles (log scale)
10^{12}
10^{11}
10^{10}
20
50
100
150
NumObj
Exec
naive
opt_soa

Reducing the flops

```
void update_collision_constraint(Vector2D *q, Vector2D *p1, Vector2D *p2, fp w) {
    // update q
    q->x += ((p1->y - p2->y) *
             (p1->y * (p2->x - q->x) + p2->y * q->x - p2->x * q->y + p1->x * (-p2->y + q->y)) *
             POW(POW(ABS(p1->x - p2->x), 2) + POW(ABS(p1->y - p2->y), 2), 2)) /
            (POW(ABS(p1->x - p2->x), 6) +
             3 * POW(ABS(p1->x - p2->x), 4) * POW(ABS(p1->y - p2->y), 2) +
             3 * POW(ABS(p1->x - p2->x), 2) * POW(ABS(p1->y - p2->y), 4) +
             POW(ABS(p1->y - p2->y), 6) +
             POW(ABS((-p1->y + q->y) * POW(ABS(p1->x - p2->x), 2) +
                      (-p1->y + q->y) * POW(ABS(p1->y - p2->y), 2) +
                      (p1->y * p2->x - p1->x * p2->y - p1->y * q->x + p2->y * q->x + p1->x * q->y -
                       p2->x * q->y) *
                          ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
                 2) +
             POW(ABS((p2->y - q->y) * POW(ABS(p1->x - p2->x), 2) +
                      (p2->y - q->y) * POW(ABS(p1->y - p2->y), 2) +
                      (-(p1->y * p2->x) + p1->x * p2->y + p1->y * q->x - p2->y * q->x -
                       p1->x * q->y + p2->x * q->y) *
                          ABS(p1->x - p2->x) * sign(-p1->x + p2->x)),
                 2) +
             POW(ABS((-p2->x + q->x) * POW(ABS(p1->x - p2->x), 2) +
                      ABS(p1->y - p2->y) * ((-p2->x + q->x) * ABS(p1->y - p2->y) +
                                            (p1->y * p2->x - p1->x * p2->y - p1->y * q->x +
                                             p2->y * q->x + p1->x * q->y - p2->x * q->y) *
```

```
void update_collision_constraint(fp *qx, fp *qy, fp *p1x, fp *p1y, fp *p2x, fp *p2y, fp w) {
    fp p1_q_x_diff = *p1x - *qx;
    fp p1_p2_x_diff = *p1x - *p2x;
    fp p2_q_x_diff = *p2x - *qx;
    fp q_p2_x_diff = *qx - *p2x;

    fp p1_p2_y_diff = *p1y - *p2y;
    fp p2_q_y_diff = *p2y - *qy;
    fp q_p1_y_diff = *qy - *p1y;

    fp p1_p2_y_diff_abs = ABS(p1_p2_y_diff);
    fp p1_p2_y_diff_sign = sign(p1_p2_y_diff);
    fp p1_p2_x_diff_pow_2 = POW(p1_p2_x_diff, 2);
    fp p1_p2_y_diff_pow_2 = POW(p1_p2_y_diff, 2);
    fp p1_p2_norm_diff = p1_p2_x_diff_pow_2 + p1_p2_y_diff_pow_2;
    fp  p1_p2_norm_diff_pow_2 = POW(p1_p2_norm_diff, 2);

    fp p1_p2_power_seq = POW(p1_p2_x_diff, 6) + 3 * POW(p1_p2_x_diff, 4) * p1_p2_y_diff_pow_2 +
                         3 * p1_p2_x_diff_pow_2 * POW(p1_p2_y_diff, 4) + POW(p1_p2_y_diff, 6);

    fp abs_p1p2x_diff_times_sign_p2p1x_diff = ABS(p1_p2_x_diff) * sign(-*p1x + *p2x);

    fp p1y_mul_p2x = *p1y * *p2x;
    fp p1x_mul_p2y = *p1x * *p2y;
    fp p1x_mul_qx = *p1x * *qx;
    fp p1y_mul_qx = *p1y * *qx;
    fp p1x_mul_qy = *p1x * *qy;
    fp p2y_mul_qx = *p2y * *qx;
    fp p2x_mul_qy = *p2x * *qy;
    fp p2y_mul_qy = *p2y * *qy;
```

Function	Flops
update_collision_constraint	1065
update_midpoint_constraint	1912

After reduction

Function	Flops
update_collision_constraint	484 (45%)
update_midpoint_constraint	595 (31%)

Reducing the flops: instruction count

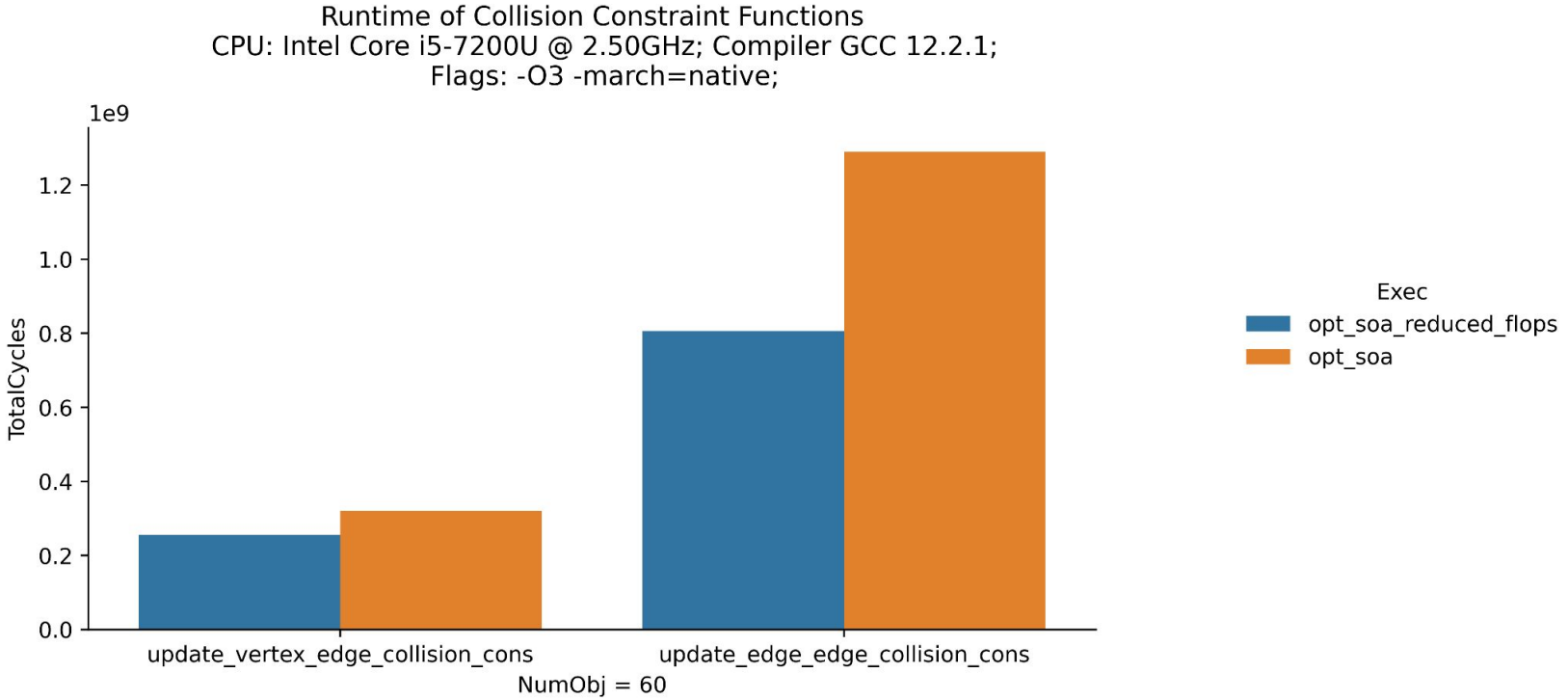
update_collision_constraint		
Compiler flag level	Before reduction	After reduction
0	4046	**1332**
1	1531	**741**
2	1128	**482**
3	1128	**386**

update_midpoint_constraint		
Compiler flag level	Before reduction	After reduction
0	6354	**1661**
1	1401	**704**
2	729	**434**
3	1014	**434**

Compiler: GCC 11.3

0: -O0 -mno-fma -fno-tree-vectorize
1: -O3 -march=native -mno-fma -fno-tree-vectorize
2: -O3 -march=native -ffast-math -fno-tree-vectorize
3: -O3 -march=native -ffast-math

Reducing the flops



Precomputing collisions

Algorithm 1 Position Based Dynamics

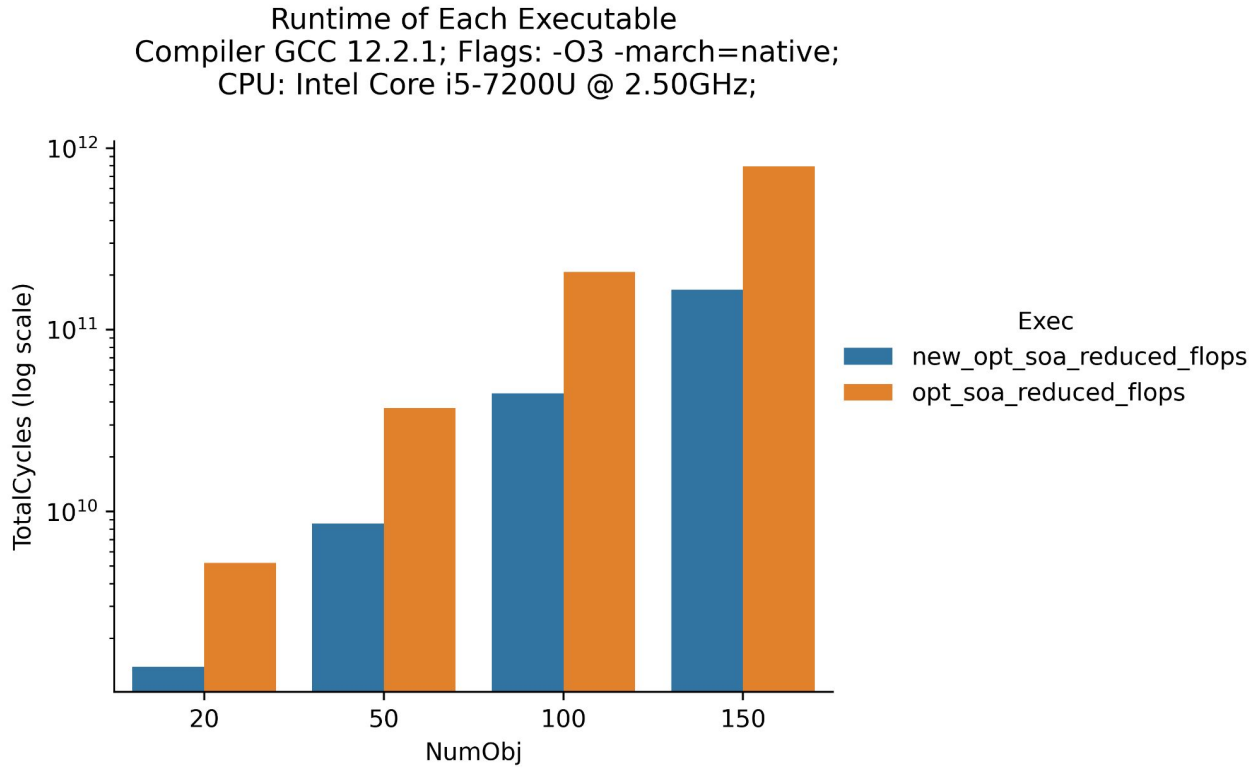
O(n²)!!!

```
1:  INITIALIZE(p_1, ..., p_N, v_1, ..., v_N)
2:  loop
3:      APPLYEXTERNALFORCES(p_1, ..., p_N, v_1, ..., v_N)
4:      INTEGRATEVELOCITIES(p_1, ..., p_N, v_1, ..., v_N)
5:      GENERATECOLLISIONCONSTRAINTS(p_1, ..., p_N)
6:      for solverIterations do
7:          PROJECTCONSTRAINTS(p_1, ..., p_N)
8:      end for
9:      UPDATESTATE(p_1, ..., p_N, v_1, ..., v_N)
10: end loop
```

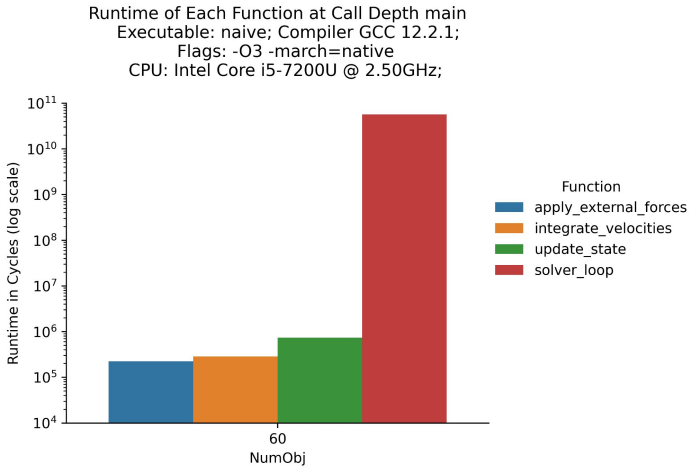
collision records stored in **lightweight buffers** upon detection

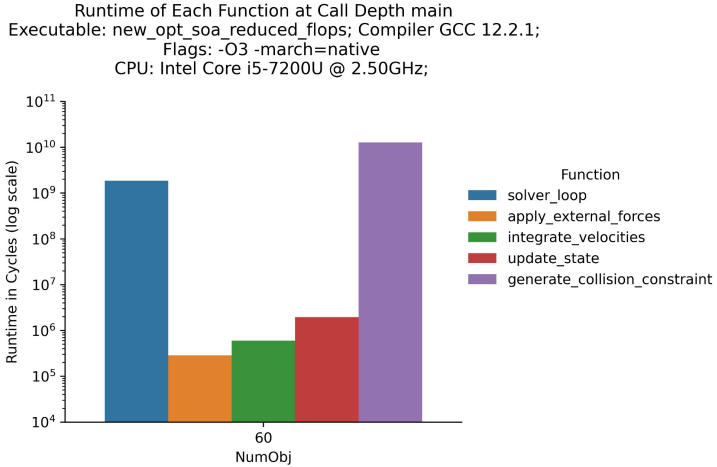
```
// --------------------
// COLLISION SPECIFICS
// --------------------

uint8_t* vert_collision_bit;
int* vert_collision_id;
uint8_t* vert_collision_edge;

uint8_t* edge_collision_bit;
int* edge_collision_id;
uint8_t* edge_collision_edge;
```

Precomputing collisions

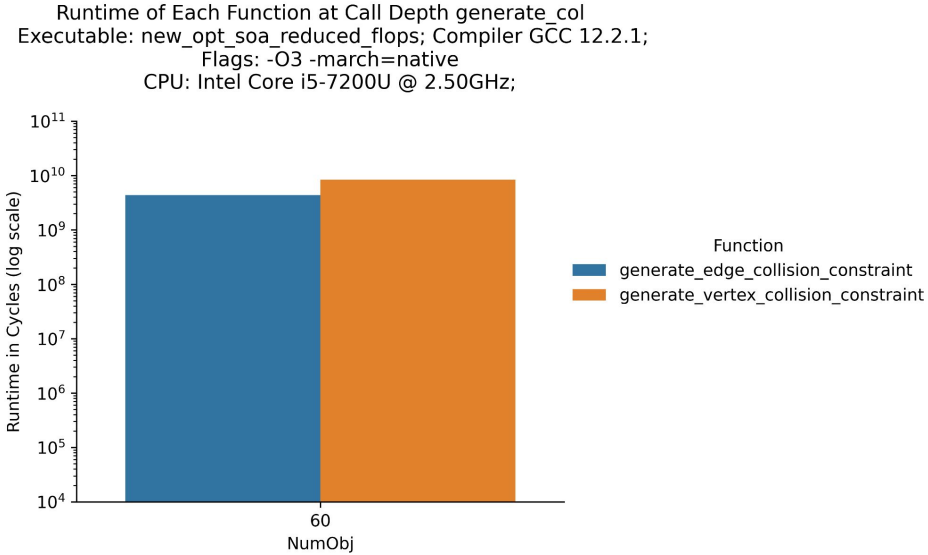


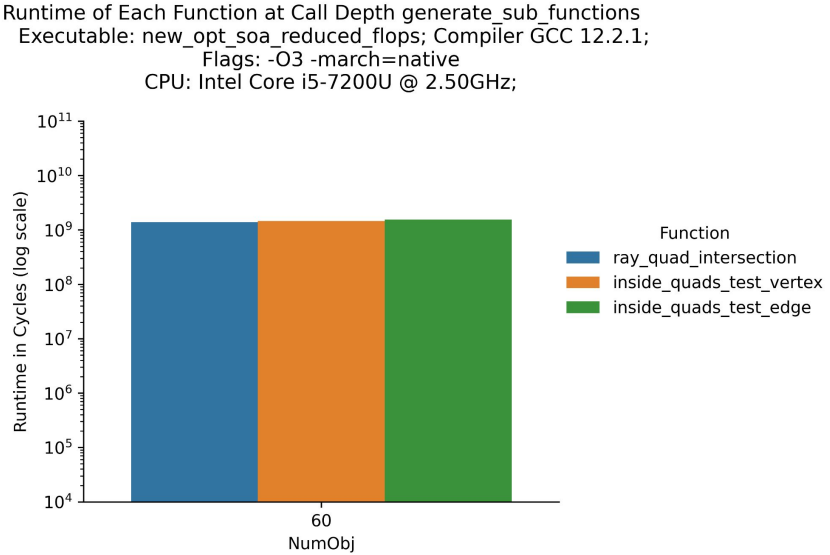
New bottleneck



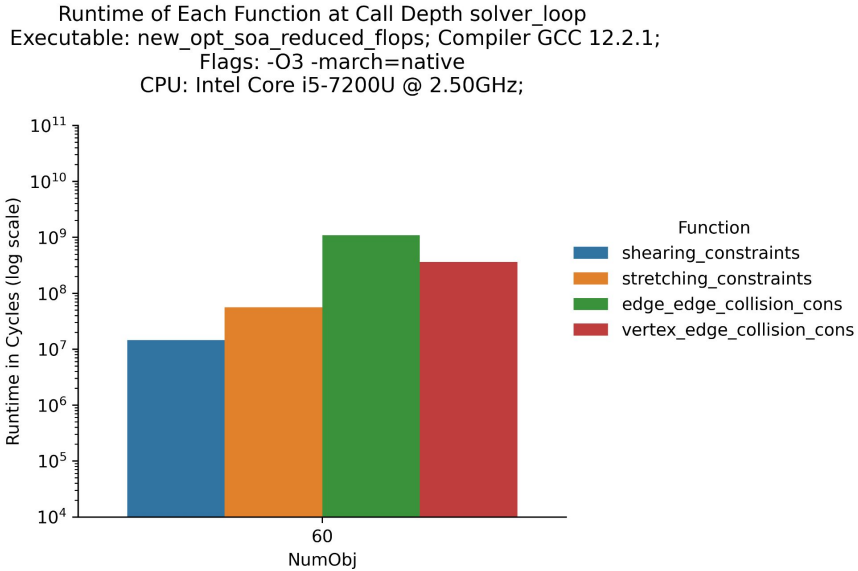


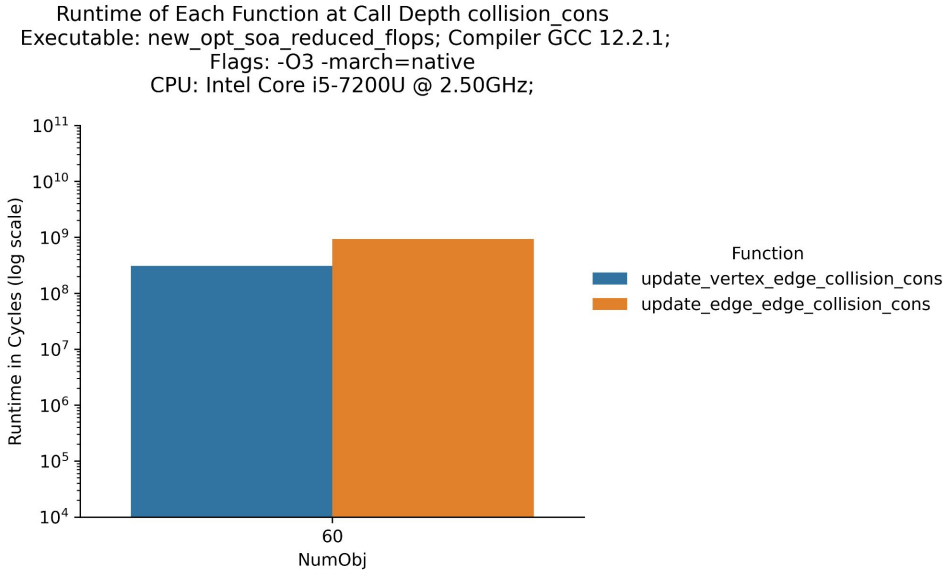
New bottlenecks: Generating Constraints Breakdown



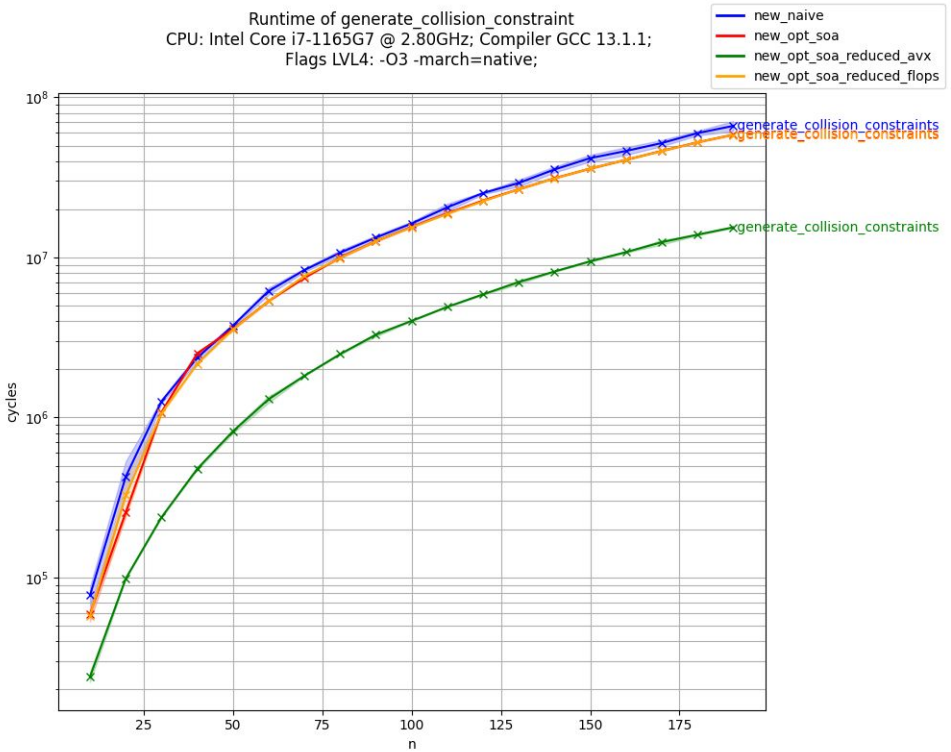


New bottlenecks: Solver Loop Breakdown



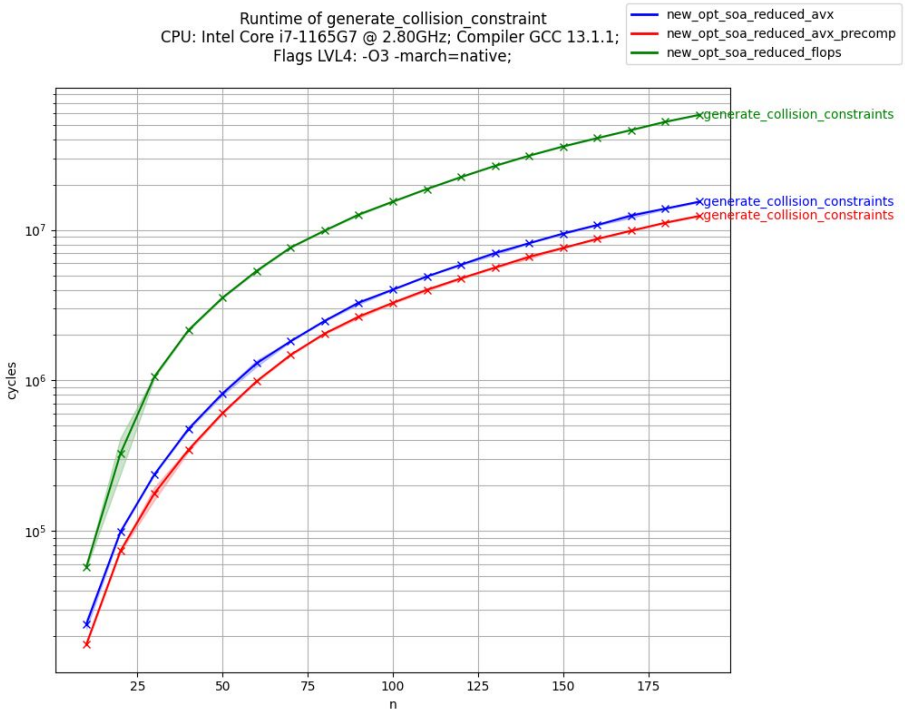


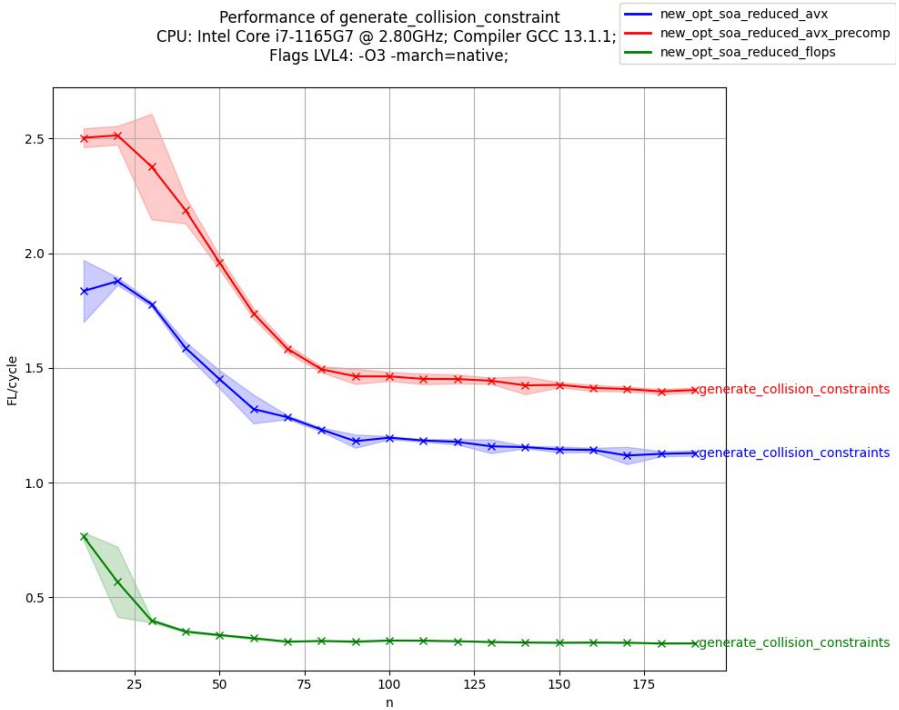
Vectorization



- Our vectorized version shows a 3.75x improvement in microbenchmarks.
- Notably faster than auto-vectorization (compiler)

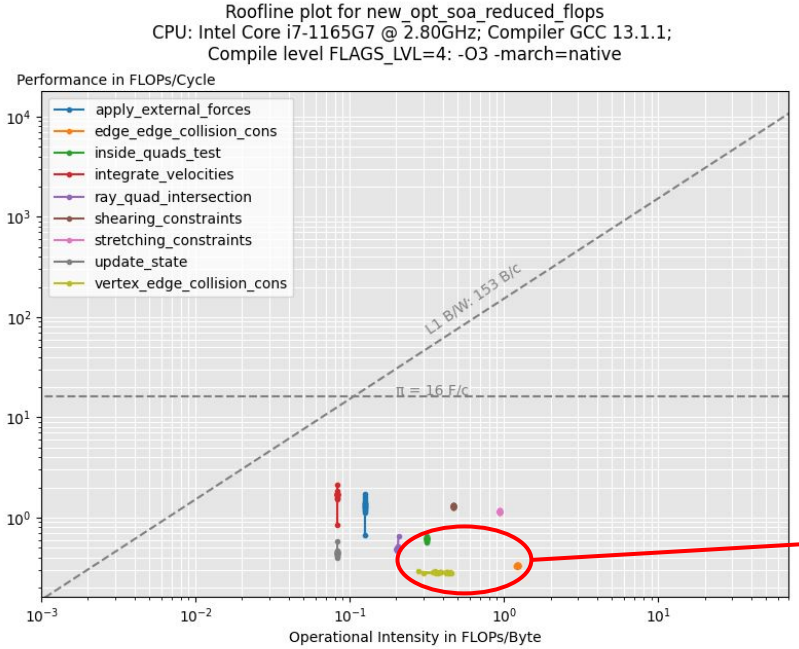
Precomputing reused values in generate_collision_constraints

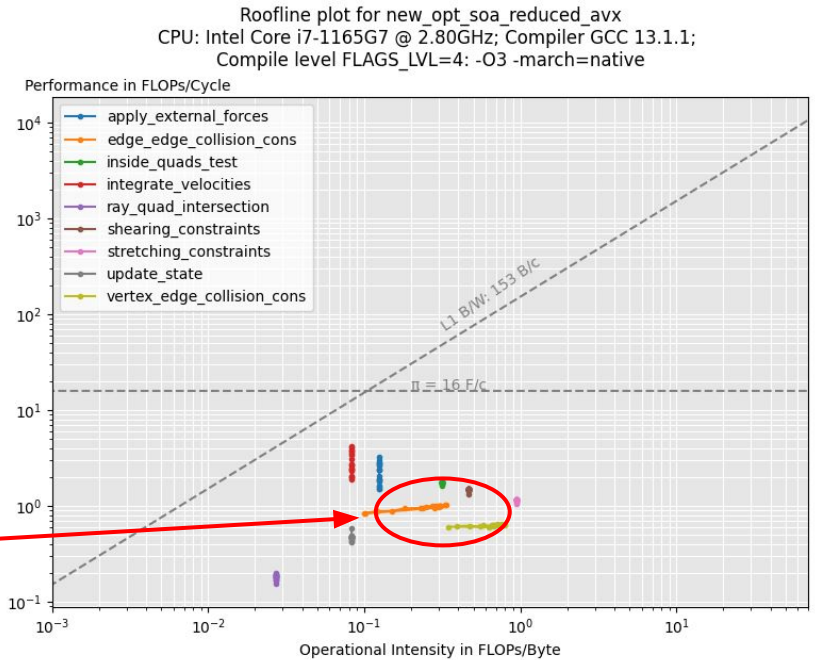




New rooflines

- We are still mostly compute-bound
- Vectorization significantly boosts performance.
- Auto-vectorization is not able to improve over scalar performance.





Runtime (in ms)

We made PBD significantly faster: a **100x speed-up** between the naive implementation and full optimizations! 🎉