Speculative Atomics
Case-study of the GPU Optimization of the Material Point Method for Graphics

Gergely Klar
UCLA Computer Graphics & Vision Laboratory
Motivation


Images ©Disney.
MPM Simulations

- Amazing new effects
- Extremely computationally demanding
  - millions of particles
  - 200 million cell grids
  - up to 30 minutes per frame
MPM Simulations

- Amazing new effects
- Extremely computationally demanding
  - millions of particles
  - 200 million cell grids
  - up to 30 minutes per frame

*GPU to the rescue!*
MPM Overview

1. Particles-to-grid
2. Grid operations
3. Particles-from-grid
4. Particle operation
Parallelizing steps

1. Particles-to-grid
2. Grid operations
3. Particles-from-grid
4. Particle operations
Parallelizing steps

1. Particles-to-grid
2. Grid operations
3. Particles-from-grid
4. Particle operations --- trivial
Parallelizing steps

1. Particles-to-grid
2. Grid operations

✓ 3. Particles-from-grid --- trivial
✓ 4. Particle operations --- trivial
Parallelizing steps

1. Particles-to-grid

✓ 2. Grid operations --- case by case
✓ 3. Particles-from-grid --- trivial
✓ 4. Particle operations --- trivial
Parallelizing steps

1. Particles-to-grid --- challenge
2. Grid operations --- case by case
3. Particles-from-grid --- trivial
4. Particle operations --- trivial
What’s wrong with *particles-to-grid*?

It is a classical *scatter* operation

- up to 10 particles per cell initially
- $5^3$ support (region of influence) per particle

$\Rightarrow$ Many race conditions, if processed particle-by-particle
How to do super fast scatter?

1. Use atomics operations!
2. Protect coalesced access!
3. Disperse the particles!
Why not *gather*?

Classical work around:

Turn *scatter* it into *gather*!

Problems:

- No constraint on particle count per cell
- Overhead
- Need to do it in every step!
Why not *gather*?

Classical work around:

Turn *scatter* it into *gather*!

Problems:

- No constraint on particle count per cell
- Overhead
- **Need to do it in every step!**
Let’s use atomics!

(and I mean *atomic instructions*)

**Pro:** Don’t need to worry about race conditions

**Con:** Can get very slow
Speculative atomics

Guaranteeing no race conditions is expensive. Atomic instructions are getting better, minimal overhead when there’s no need to block.

**Idea:** Don’t try to eliminate all race conditions, but reduce their likelihood and use atomic instructions.
Bad case
Bad case
Worst case
Best case
Best case
How to get the best case all the time?

Concurrent threads need to process particles that

● are from different cells
● are consecutive in memory
● access consecutive grid cells
Bad idea: Indirect indexing

processParticle(p[particleIndexShuffler(threadIdx)])

Problem: particles processed in parallel are spread out in memory → can not use coalesced memory access → poor performance

Solution: need to physically shuffle particles
Blocked layout

Particles in memory

Particles in space
Blocked layout in action
Blocked layout

1. Assign cell ID to each particle by the grid cell it is in. cell IDs define groups.
2. Sort particle IDs by cell IDs
3. Rearrange particles by picking the 1\textsuperscript{st} from each group, then the 2\textsuperscript{nd}, etc.
kernelComputeCellID // per particle

cellX = floor(x[pID]/dx)
cellY = floor(y[pID]/dx)
cellZ = floor(z[pID]/dx)
cellID[pID] = cellX +
    cellY*GRID_WIDTH +
    cellZ*GRID_WIDTH*GRID_HEIGHT
sequence(pIDs)
sort_by_key(cellIDs, pIDs)
exclusive_scan_by_key(cellIDs, constant(1), newIDs)
sort_by_key(newIDs, permutation(zip(<all particle data>), pIDs))
Re-blocking

No need to rearrange the particles in every step, but

● Goal of MPM is to simulate deformations and fracturing
● Particles can move around a lot
● Monitor scatter performance, re-block if drops below threshold
Results

Relative speeds compared to the CPU implementation

CPU MPM: 100%

GPU Naïve Atomics: 9.8%
GPU Gather: 4.9%
GPU Blocked Layout: 2.2%
## Results

Relative speed-ups compared to other methods

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU Naïve</th>
<th>GPU Gather</th>
<th>GPU B. Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>1x</td>
<td>10.20x</td>
<td>20.41x</td>
<td>45.45x</td>
</tr>
<tr>
<td><strong>GPU Naïve</strong></td>
<td></td>
<td>1x</td>
<td>2.00x</td>
<td>4.45x</td>
</tr>
<tr>
<td><strong>GPU Gather</strong></td>
<td></td>
<td></td>
<td>1x</td>
<td>2.23x</td>
</tr>
<tr>
<td><strong>GPU B. Layout</strong></td>
<td></td>
<td></td>
<td></td>
<td>1x</td>
</tr>
</tbody>
</table>
Summary

● Speculative Atomics:
  ○ The overhead of unnecessary atomic instructions is lower than the overhead of complicated gather transforms
● Blocked layout:
  ○ Special arrangement of particles to minimize the number of race conditions
● Not just for MPM:
  ○ Can be adopted to other particles-and-grids simulations as well
Takeaway

1. Don’t be afraid to use atomic operations!
2. Use coalesced access at all costs!
3. Spread out the particles (but not too far)!
Questions?
Thank you!