GPU Optimizations of Material Point Method and Collision Detection

Xinlei Wang, 王鑫磊
浙江大学
Material Point Method

- Fluid
  - Smoothed-Particle Hydrodynamics
  - Grid-based Methods

- Solid
  - Finite Element Method
  - Finite Difference Method

- Material Point Method
  - large deformation, complex topology changes
  - multi-material & multiphase coupling
  - (self) collision handling
MPM Pipeline Overview

- **Lagrangian material particles**
  - $m_p^n$ $v_p^n$ $x_p^n$

- **Eulerian Cartesian grids**
  - $m_i^n$ $p_i^n$

- **Advection**
  - $x_p^n$ $v_p^{n+1} F_p^{n+1}$

- **Maintain Structures**
  - Particle: Sort & Order
  - Sparse Grid: Generate Sparse Blocks
  - Particle – Grid Mapping

- **Rasterize**
  - Material Stress Computation
  - **Particle-to-Grid Transfer** (mass, momentum, etc.)

- **Time Integration**
  - Explicit: $v_i^{n+1} = (p_i^n + \delta t * f_{ext})/m_i^n$
  - Implicit: Solve for $v_i^{n+1}$

- **Resample**
  - Grid-to-Particle Transfer (velocity)

- **Advection**
  - Update Particle Attributes (position, deformation gradient, etc.)

- **Transfers**
  - Particle to grid
  - Grid to particle

- **Time Integration**
  - explicit
  - implicit

- **Up to 90%**
Performance is the Solution

• “dx gap”
  • a gap between adjacent models when colliding
  • increase grid resolution => more particles to achieve equal magnitude

• CFL Condition
  • for simulation stability and collision handling
  • more time steps per frame => more work to compute a frame

• Performance is the key!
Hardware Friendly Solutions

• MLS MPM

• Async MPM

• GVDB

• Warp for Cell

• Bottleneck: Particle-to-Grid Transfer
The Alternative of Transfer

<table>
<thead>
<tr>
<th>lane id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>node id</td>
<td>n</td>
<td>n+1</td>
<td>n+1</td>
<td>n+1</td>
<td>n+1</td>
<td>n+2</td>
<td>n+2</td>
<td>n+3</td>
</tr>
<tr>
<td>boundary mark</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>region interval</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **warp intrinsics**
  - **ballot clz**
  - **shfl**

- **iteration 0, stride 1**
  - attribute mass
    - mass sum
- **iteration 1, stride 2**
  - mass sum

- **shared memory**
  - node n
  - node n+1
  - node n+2
  - node n+3
Comparison

Optimized Scatter

• No auxiliary structures or memory
• Uniform workload for each thread
• Very few ‘atomicAdd’ write conflicts

Gather

• Additional particle list for each grid node
• Divergent workload
• No write-conflicts at all
• vs. FLIP [Gao et al. 2017]
  • CPU-based, Gather-style
  • ~16X Speed-up
• vs. MLS [Hu et al. 2018]
  • CPU-based, Scatter-style
  • ~8X Speed-up
• vs. Naïve Scatter
  • GPU-based, Scatter-style
  • ~10~24X Speed-up
• vs. GVDB [Wu et al. 2018]
  • GPU-based, Gather-style
  • ~7~15X Speed-up

CPU: 18-core Intel Xeon Gold 6140, ¥16000
GPU: Nvidia Titan XP, ¥8000

Performance Benchmarks
Fundamental Implementation Choices

• Data Structure for Particles
  • Arrays in the SoA (Structure of Array) layout

• Data Structure for Space
  • Perceptionally a sparse uniform grid
  • Support efficient interpolation operations
  • GSPGrid vs. GVDB

• Sort
  • Radix sort vs. Histogram sort
Performance Factors

- When the number of particles is fixed,
  - \( ppc \uparrow, \ node \downarrow, \ performance \uparrow \)

- Particle distribution doesn’t matter much
- The number of particles matters
Delayed Ordering Speedup

![Bar Chart]

- Mapping
- Stress
- P2G
- Solver
- G2P
- Sorting
- Others

[Reorder] [No Reorder]
Delayed Ordering

• Particle Attributes Classification

• By Perception
  • Intrinsics: Mass, Physical Property (Constitutive Model, etc.)
  • Extrinsics: Position, Velocity, Deformation Gradient, Affine Velocity Field (or Velocity Gradient)

• By Access (Write/Read) Frequency
  • Mass: remains static after initialized, read once per timestep
  • Position: maintained after each timestep,
  • Everything else (Velocity, Deformation Gradient, Affine Velocity Field, etc.)
Ordering Strategy

particle index

step $n-1$

step $n$

step $n+1$

particle attribute

$m_0^n$ $m_1^n$ $m_2^n$ $m_3^n$ $m_4^n$ $m_5^n$ $m_6^n$ $m_7^n$

$v_3^n$ $v_4^n$ $v_1^n$ $v_2^n$ $v_6^n$ $v_0^n$ $v_7^n$ $v_5^n$

$x_3^n$ $x_4^n$ $x_1^n$ $x_2^n$ $x_6^n$ $x_0^n$ $x_7^n$ $x_5^n$

$m_0^n$ $m_1^n$ $m_2^n$ $m_3^n$ $m_4^n$ $m_5^n$ $m_6^n$ $m_7^n$

$v_3^n$ $v_4^n$ $v_5^n$ $v_1^n$ $v_2^n$ $v_6^n$ $v_0^n$ $v_7^n$ $v_5^n$

$x_3^n$ $x_4^n$ $x_5^n$ $x_1^n$ $x_2^n$ $x_6^n$ $x_0^n$ $x_7^n$ $x_6^n$

$m_0^n$ $m_1^n$ $m_2^n$ $m_3^n$ $m_4^n$ $m_5^n$ $m_6^n$ $m_7^n$

$v_3^n$ $v_4^n$ $v_5^n$ $v_1^n$ $v_2^n$ $v_6^n$ $v_0^n$ $v_7^n$ $v_6^n$

$x_7^n$ $x_1^n$ $x_6^n$ $x_4^n$ $x_5^n$ $x_2^n$ $x_0^n$ $x_3^n$
## Ordering Strategy

Access times per-particle per-timestep

### Reorder Everything

<table>
<thead>
<tr>
<th>Particle Attribute (Dimension)</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>contiguous</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>arbitrary</td>
<td>0</td>
<td>1+1</td>
</tr>
<tr>
<td>contiguous</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Delayed Ordering

<table>
<thead>
<tr>
<th>Particle Attribute (Dimension)</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>contiguous</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>arbitrary</td>
<td>1</td>
<td>1+1</td>
</tr>
<tr>
<td>contiguous</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

...
Delayed Ordering Speedup
Summary:

• GPU MPM pipeline
  • efficient, extensible, cross-platform
  • support multiple-materials
  • https://github.com/kuiwuchn/GPUMPM

• What’s next?
  • Multi-GPU MPM
  • Distributed GMPM
Collision Detection

• Broad-phase Collision Detection
• Look for AABB bounding box intersections
• Typical memory-bound CUDA kernels!
BVH (Bounding Volume Hierarchy) Construction

• BVH Construction
  • [2012 Karras] builds all nodes in parallel
  • [2014 Apetrei] builds & refits in one iteration

• BVH Stackless Traversal
  • [2007 Damkjaer] depth-first order traversal using escape index

Linear BVH built on top of primitives sorted by their Morton codes
Stackless BVH Traversal

- **BVH Construction**
  - [2012 Karras] builds all nodes in parallel
  - [2014 Apetrei] builds & refits in one iteration

- **BVH Stackless Traversal**
  - [2007 Damkjaer] depth-first order traversal using escape index

Depth-first order traversal track of *Primitive-1* assuming it collides with all the other primitives
BVH-based Collision Detection

- Full traversal of the internal nodes
  - Original BVH: 4 2 1 0 3 6 5
  - Ordered BVH: 0 1 2 3 4 5 6

- How to compute BVH order
  - Calculate the LCL-value of each leaf node
  - Compute prefix sums of LCL-values
  - Assign the indices from LCA from top to bottom
Effectiveness of ordering

- **Without ordering**
  - L2 Cache Hit Rate (L1 Reads)
    - 88%
  - Global Load L2 Transactions/Access
    - 31.7
  - Maximum Divergence
    - 99.9%

- **With ordering**
  - L2 Cache Hit Rate (L1 Reads)
    - 92%
  - Global Load L2 Transactions/Access
    - 23.4
  - Maximum Divergence
    - 65.7%

2-3x speedup!

- The overhead of histogram sort is low (~1ms)
Thanks!

https://github.com/littlemine
Xinlei Wang, 王鑫磊
Other Useful Engineering Tips

• For Performance:
  • SoA memory layout
  • Per-material computation, separate material properties from particle attributes

• For Code Reusability:
  • Entity-Component System
    • Particle extrinsics formulation relies on certain components (MLS/non-MLS, PIC/FLIP/APIC)
  • Functional Programming
    • Implicit Time Integration involves lots of similar grid operations
    • Transfer schemes can be formulated by various submodules (kernel, transfer method)
    • Easier to make task parallel