GPU Optimizations of Material Point Method and Collision Detection

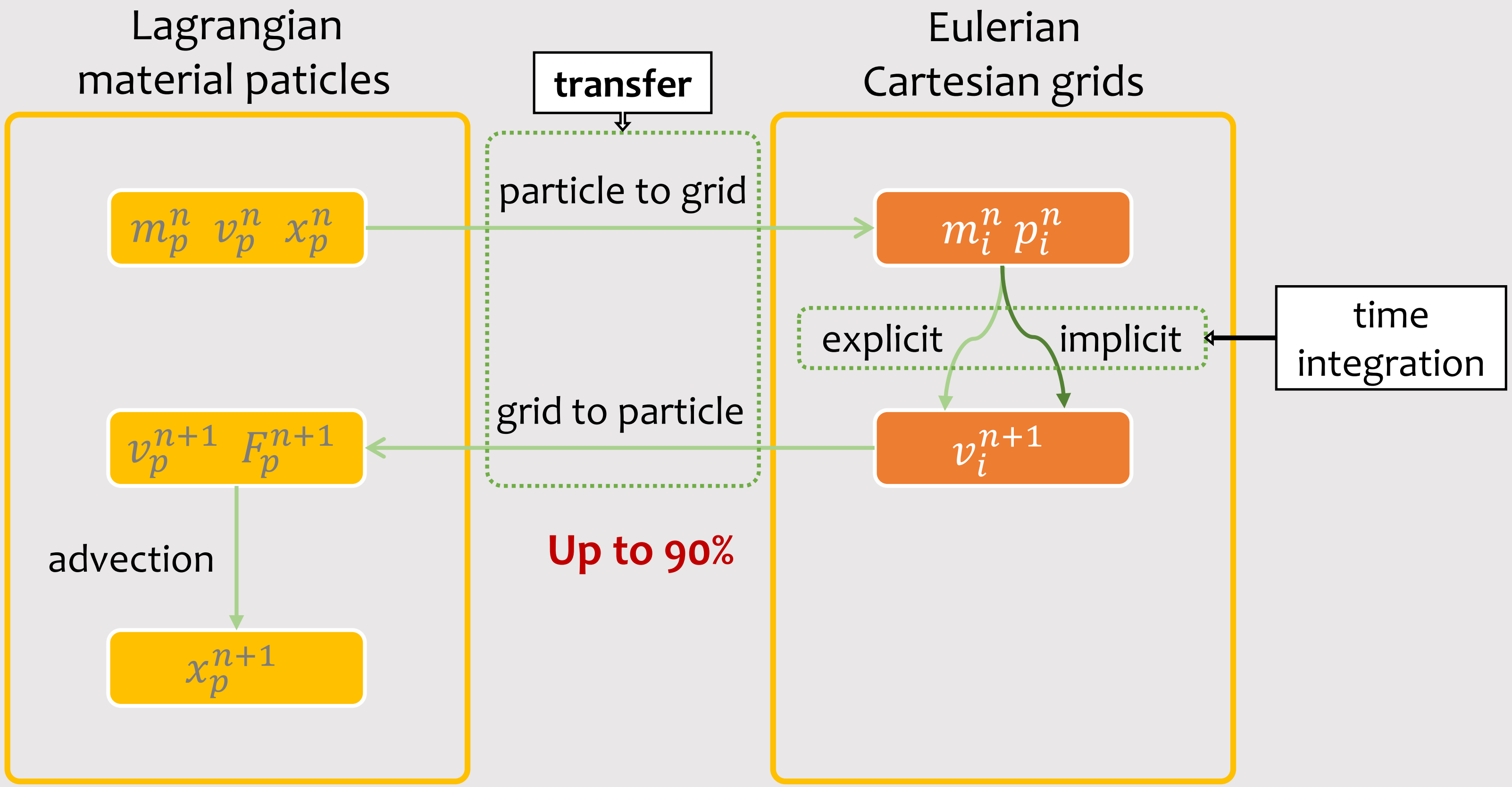
Xinlei Wang, 王鑫磊
浙江大学

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



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}

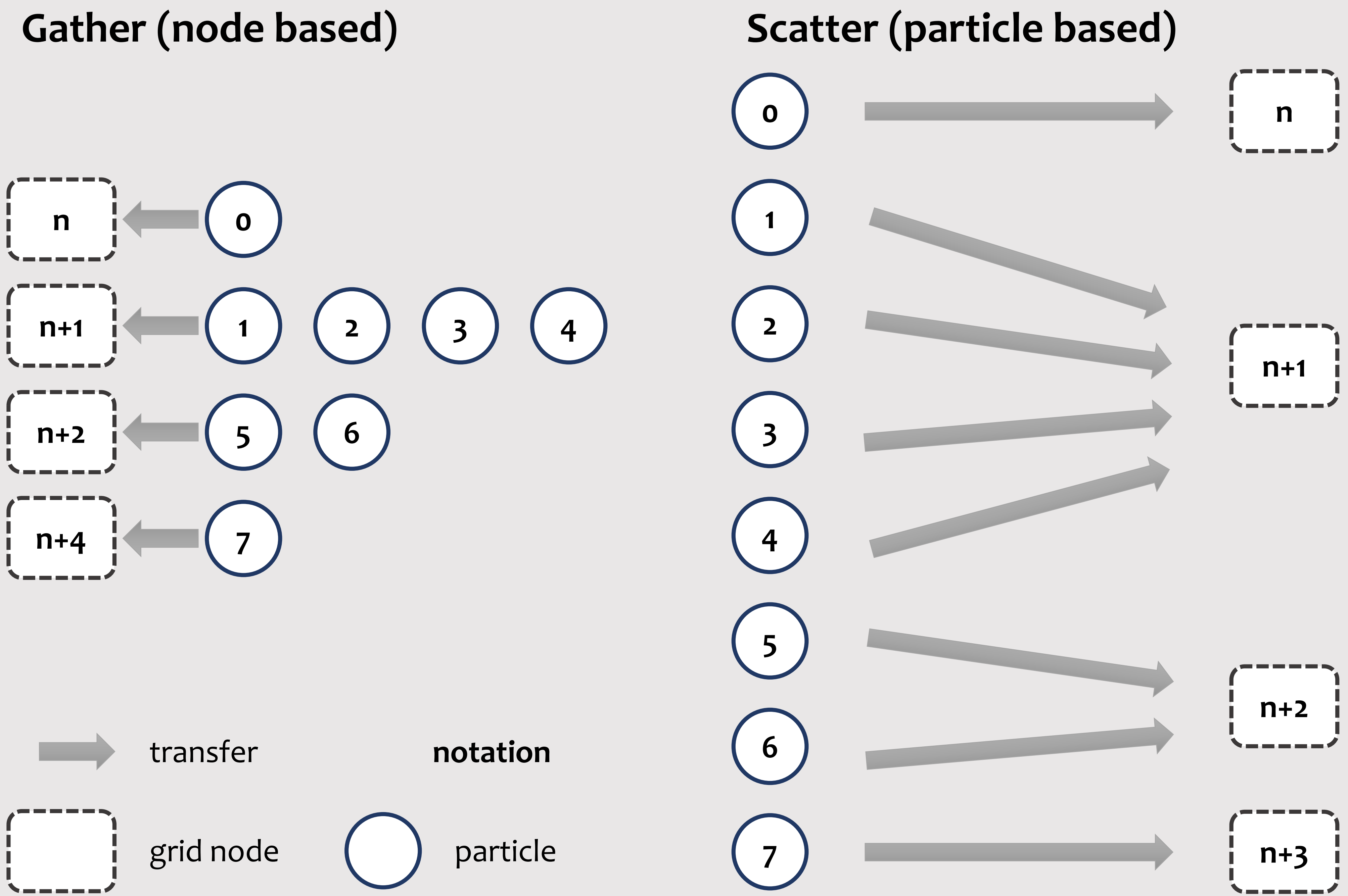
Resample
- Grid-to-Particle Transfer (velocity)

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

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!


Gather (node based)
Scatter (particle based)
n
n+1
n+2
n+4
0
1
2
3
4
5
6
7
n
n+1
n+2
n+3
transfer
notation
grid node
particle

Hardware Friendly Solutions

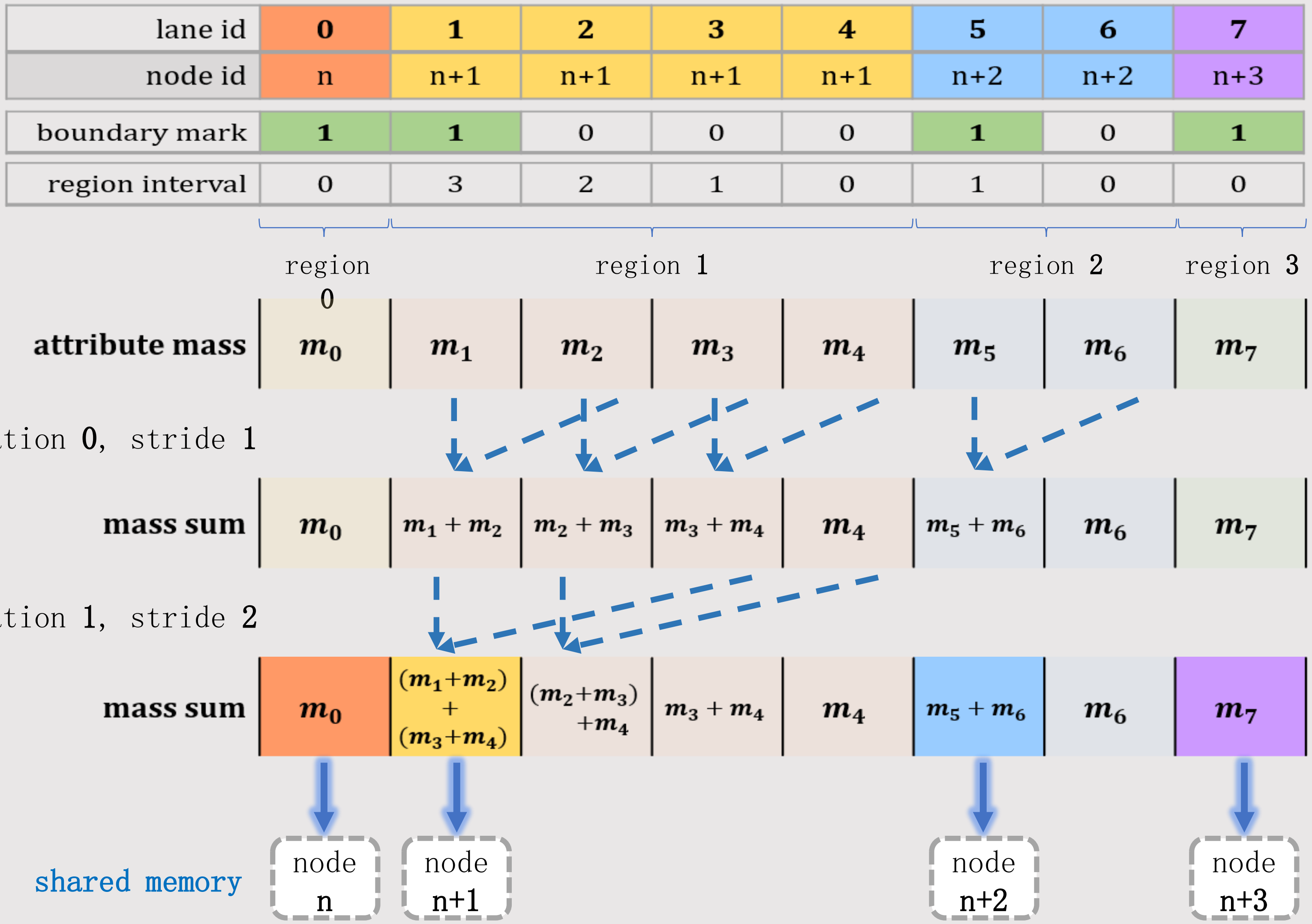
- MLS MPM
  - [2018 SIGGRAPH, Hu, et al.] A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling
- Async MPM
  - [2018 SCA, Fang, et al.] A Temporally Adaptive Material Point Method with Regional Time Stepping
- GVDB
  - [2018 EG, Wu, et al.] Fast Fluid Simulations with Sparse Volumes on the GPU
- Warp for Cell
  - [2017 GTC, Museth, et al.] Blasting Sand with NVIDIA CUDA: MPM Sand Simulation for VFX
  - http://on-demand.gputechconf.com/gtc/2017/video/s7298-ken-museth-blasting-sand-with-nvidia-cuda-mpm-sand-simulation-for-vfx.mp4
- Bottleneck: Particle-to-Grid Transfer

The Alternative of Transfer

warp intrinsics

ballot
clz

shfl



Comparison

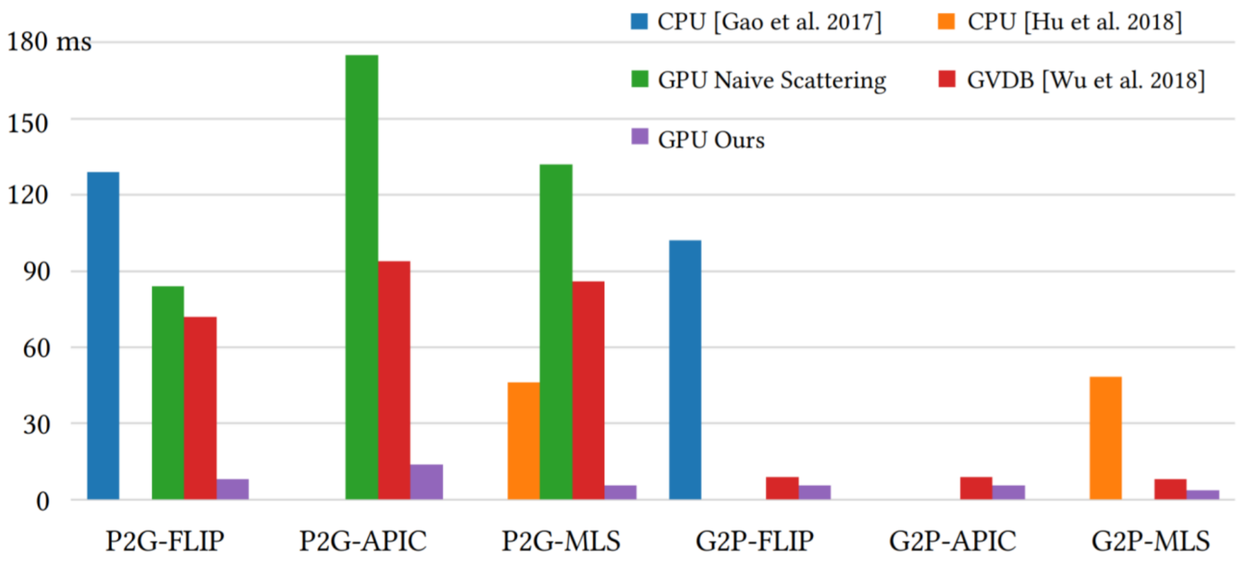
Optimized Scatter

- 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

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



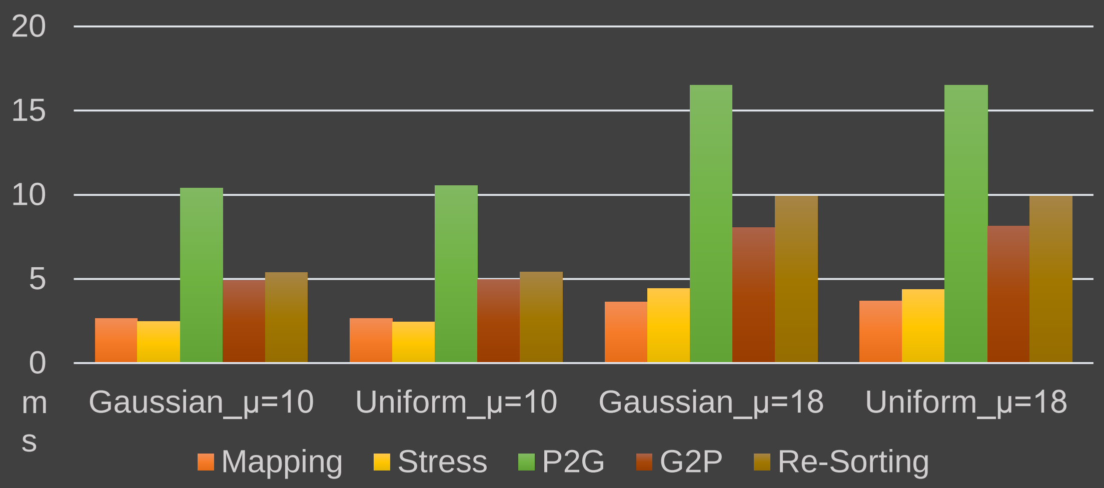
- vs. FLIP [Gao et al. 2017]
  - CPU-based, Gather-style
  - ~16X Speed-up
- vs. MLS [Hu et al. 2018]
  - CPU-based, Scatter-style
  - ~8X Speed-up
- vs. Naïve Scatter
  - GPU-based, Scatter-style
  - ~10~24X Speed-up
- vs. GVDB [Wu et al. 2018]
  - GPU-based, Gather-style
  - ~ 7~15X Speed-up

Performance Benchmarks

Fundamental Implementation Choices

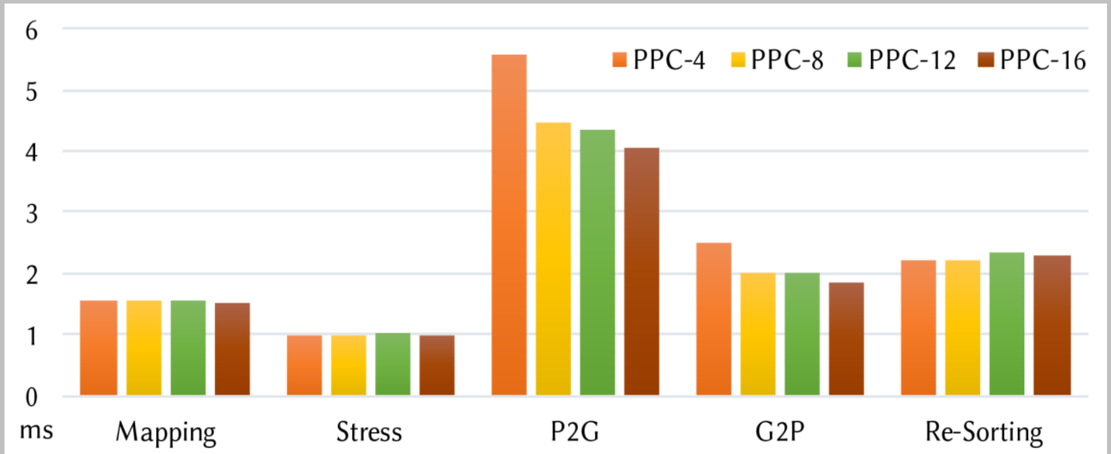
- Data Structure for Particles
  - Arrays in the SoA (Structure of Array) layout
- Data Structure for Space
  - Perceptionally a sparse uniform grid
  - Support efficient interpolation operations
  - GSPGrid vs. GVDB
- Sort
  - Radix sort vs. Histogram sort

Performance Factors

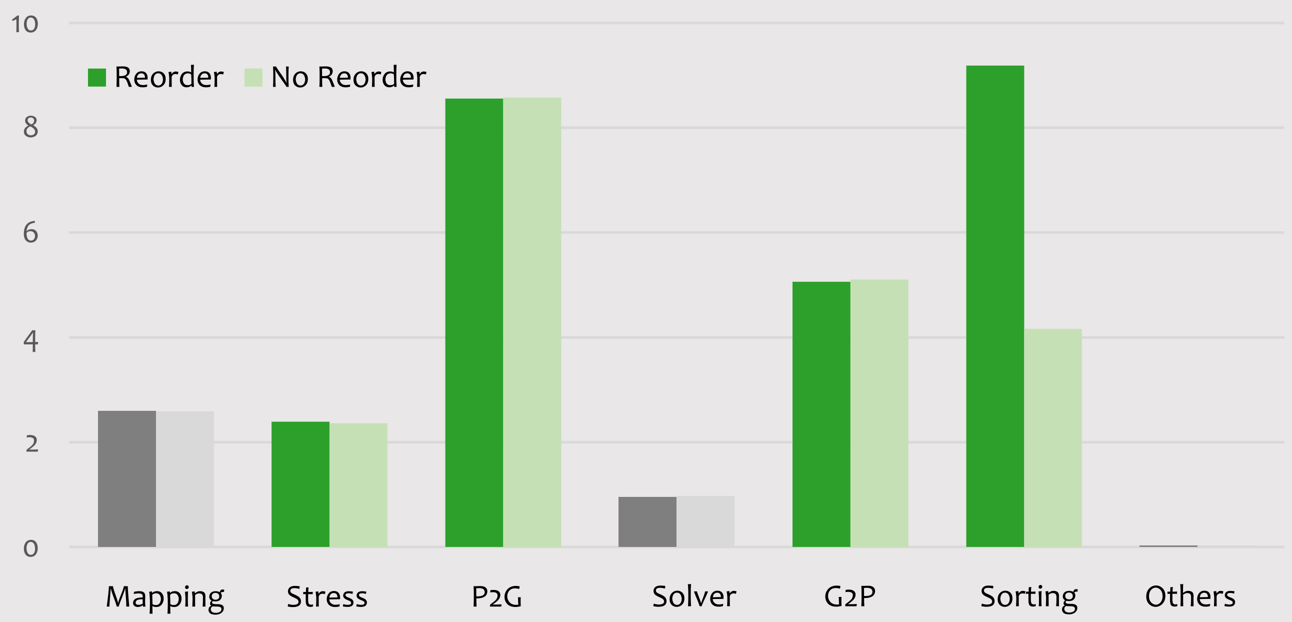


- Particle distribution doesn't matter much
- The number of particles matters

- When the number of particles is fixed,
- ppc↑, node↓, performance↑



Delayed Ordering Speedup

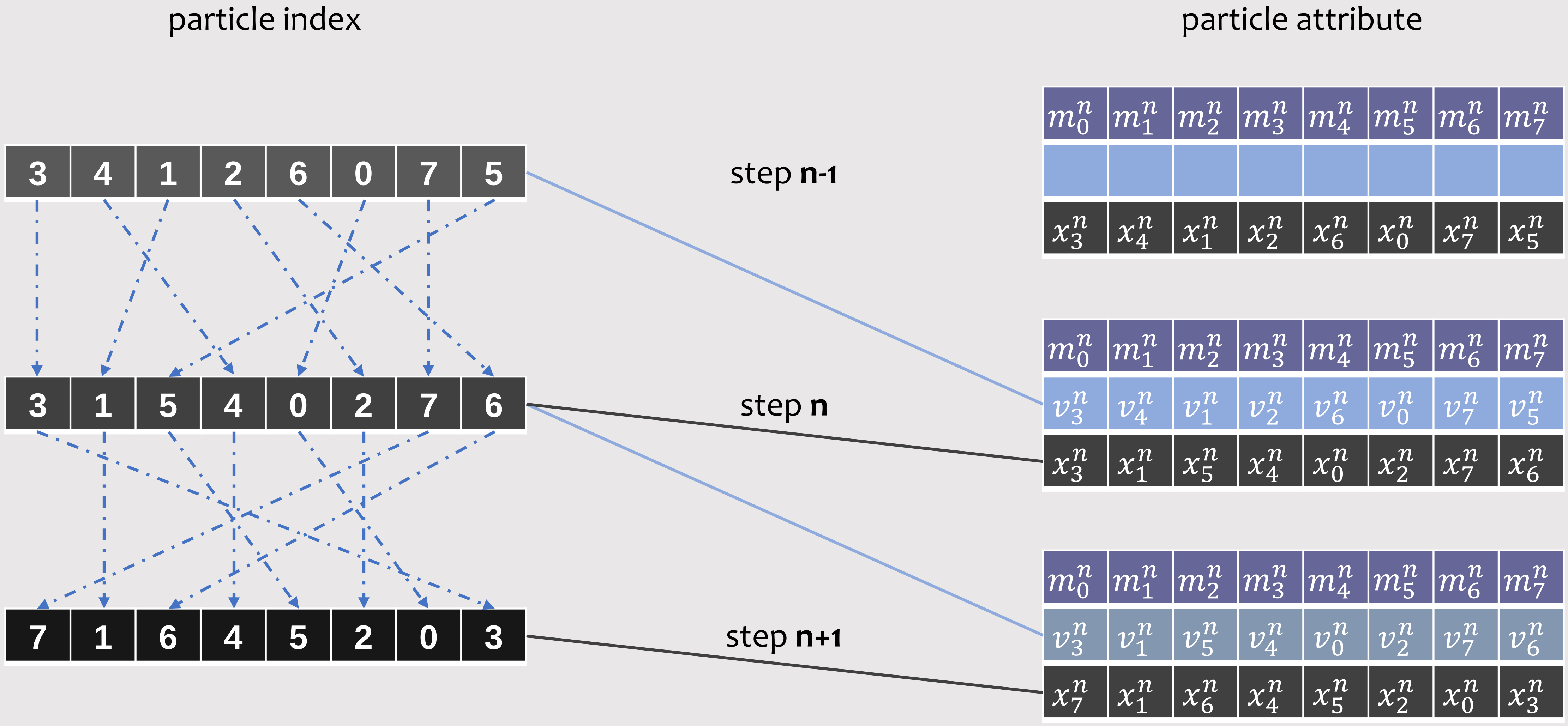


Delayed Ordering

- 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



Ordering Strategy

Access times per-particle per-timestep

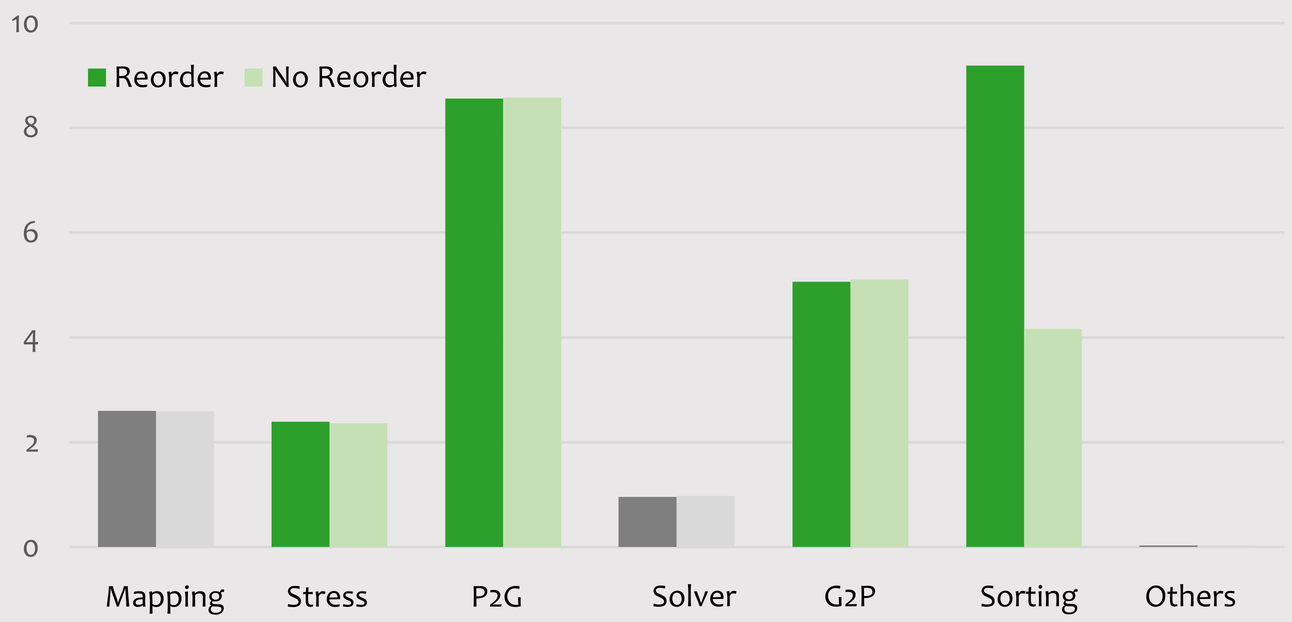
Reorder Everything

Particle Attribute (Dimension)	Read		Write	
	arbitrary	contiguous	arbitrary	contiguous
mass (1)	1	1	0	1
position (d)	1	3	0	1+1
velocity (d)	1	1	0	1+1
deformation gradient (d*d)	1	1	0	1+1
…	…		…	

Delayed Ordering

Particle Attribute (Dimension)	Read		Write	
	arbitrary	contiguous	arbitrary	contiguous
mass (1)	1	0	0	0
position (d)	1	3	0	1+1
velocity (d)	1	0	0	1
deformation gradient (d*d)	0	1	0	1
…	…		…	

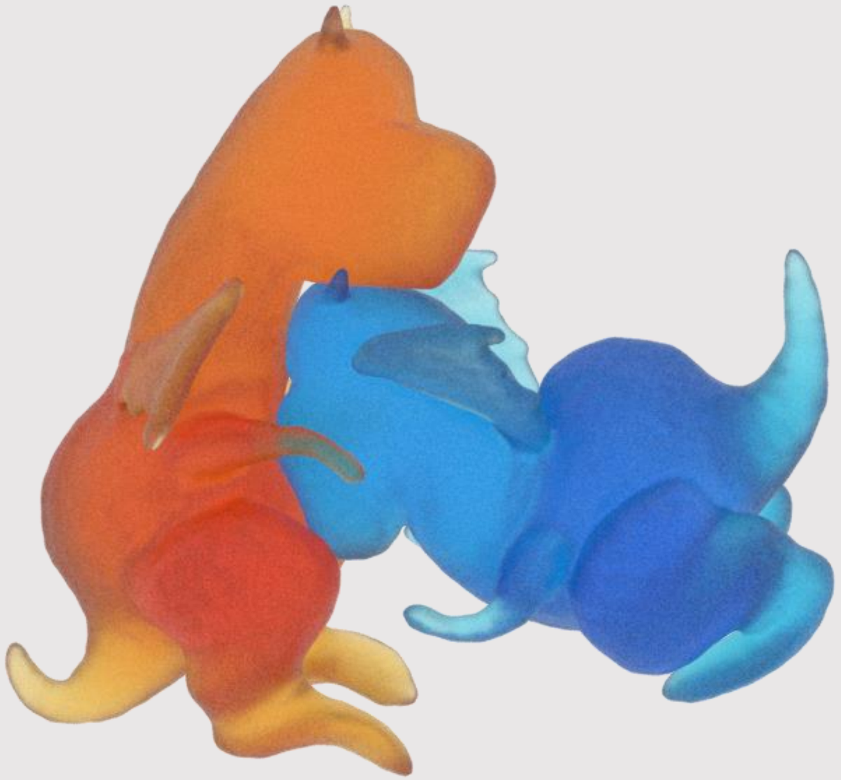
Delayed Ordering Speedup

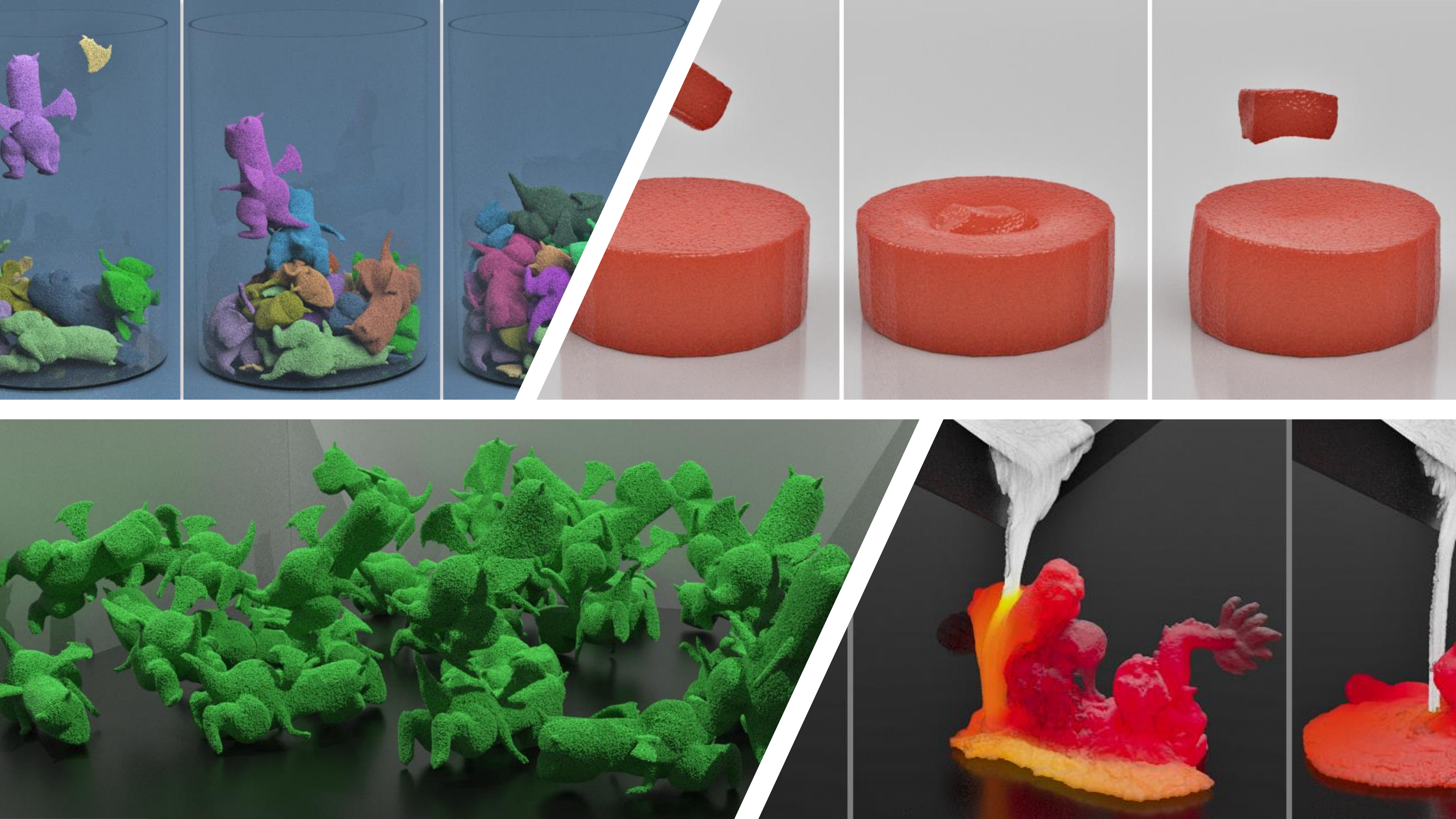


Summary:

- 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

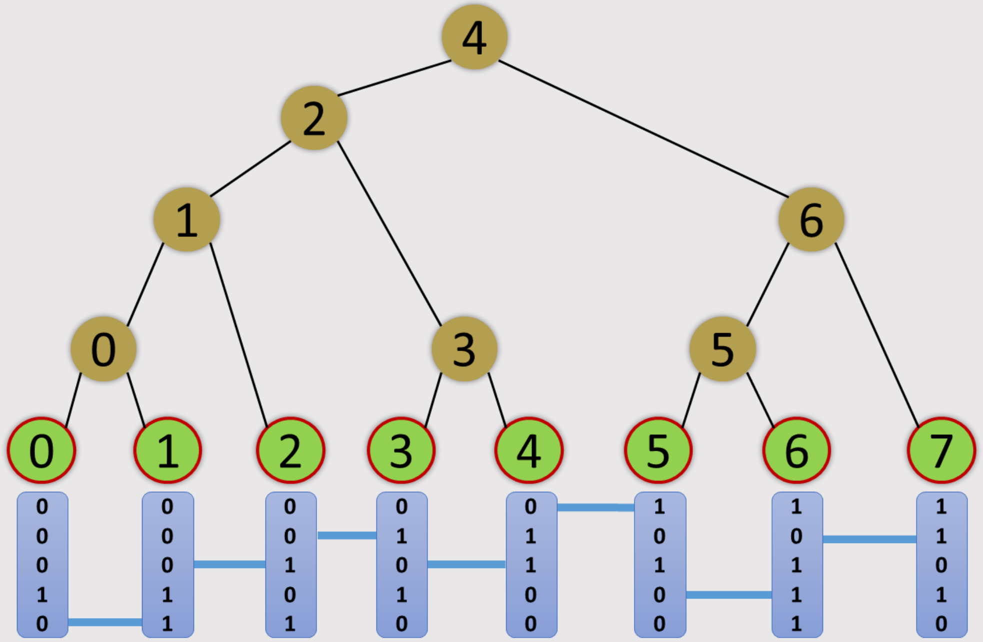
- Broad-phase Collision Detection
- Look for AABB bounding box intersections

- 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

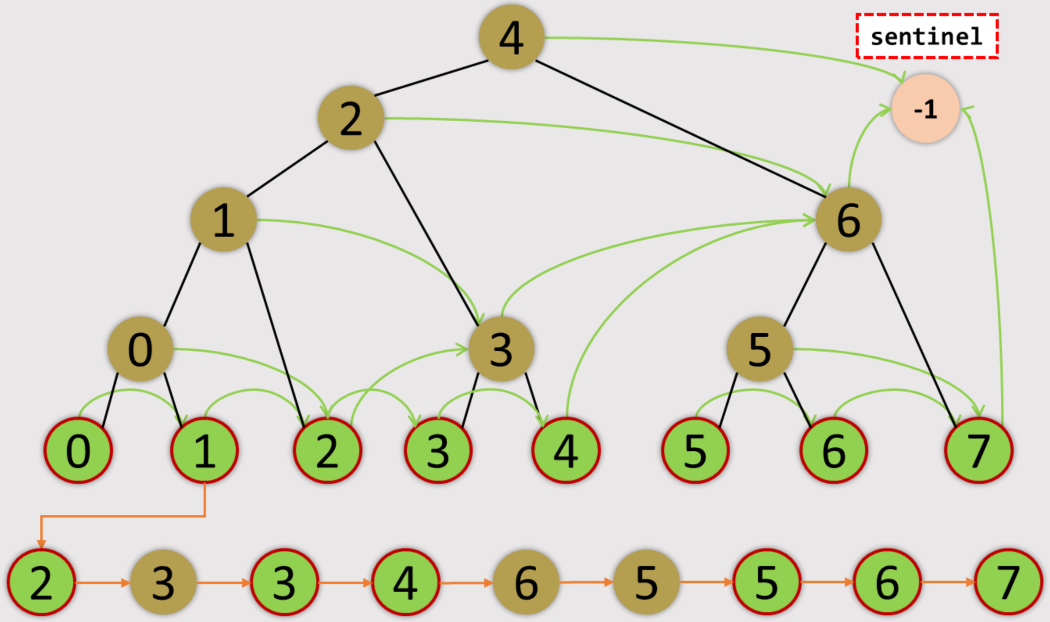


Linear BVH built on top of primitives sorted by their Morton codes

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

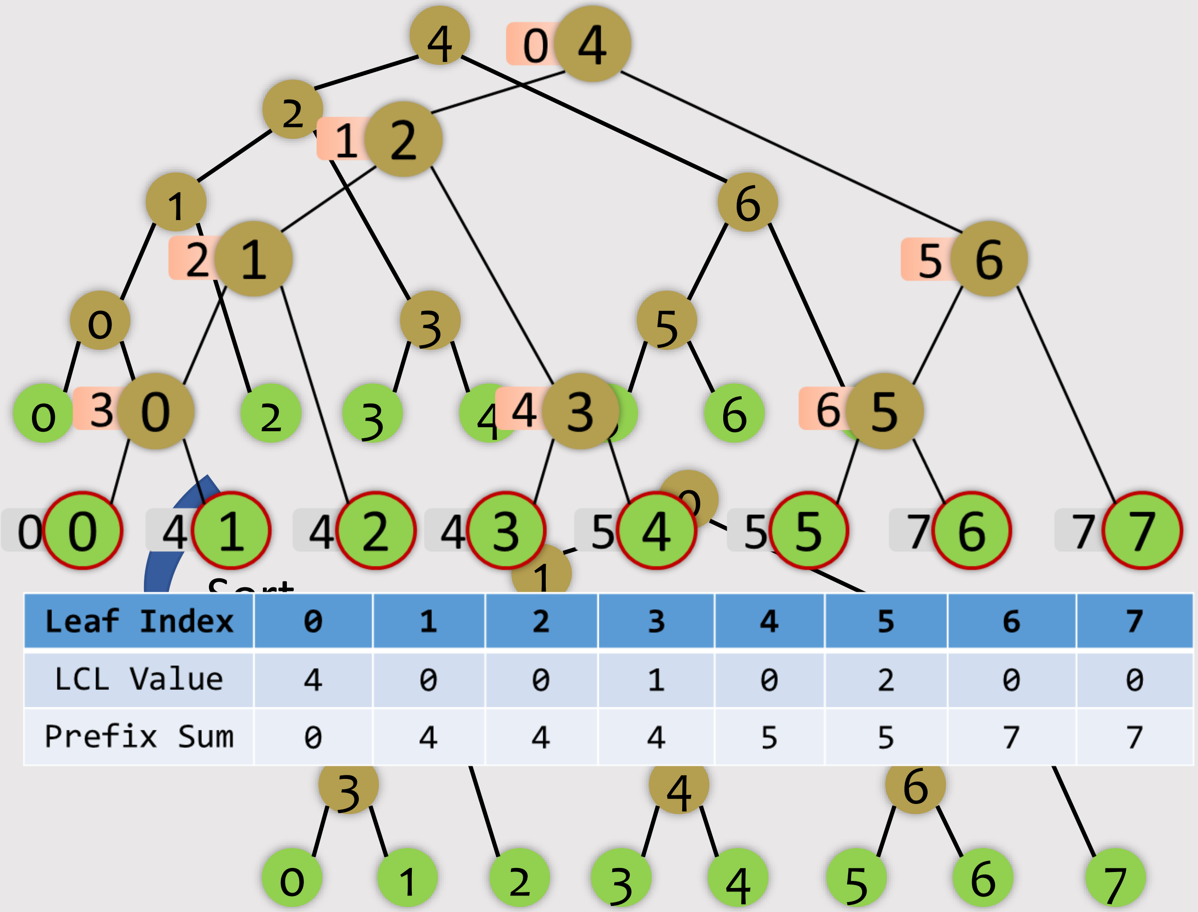


Depth-first order traversal track of *Primitive-1* assuming it collides with all the other primitives

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



Leaf Index	0	1	2	3	4	5	6	7
LCL Value	4	0	0	1	0	2	0	0
Prefix Sum	0	4	4	4	5	5	7	7

Effectiveness of ordering

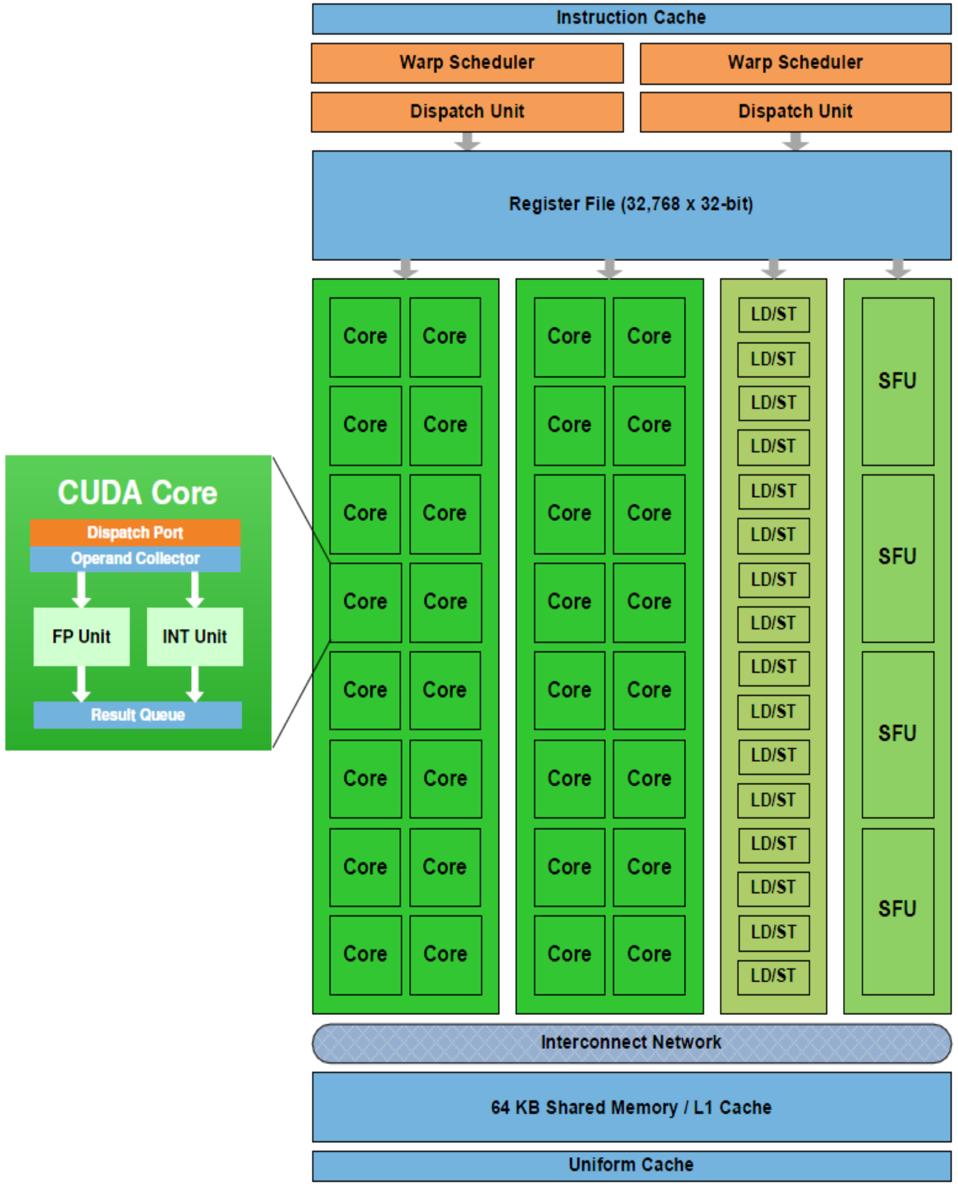
- **Without ordering**
- L2 Cache Hit Rate (L1 Reads)
  - 88%
- Global Load L2 Transactions/Access
  - 31.7
- Maximum Divergence
  - 99.9%

2~3x speedup !

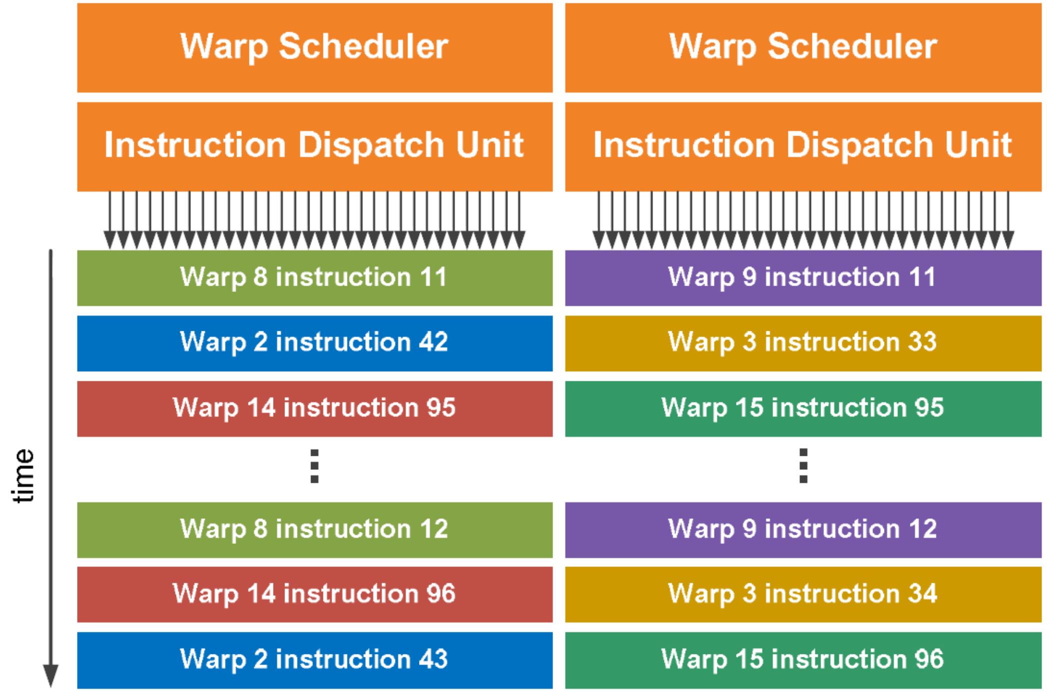
- **With ordering**
- L2 Cache Hit Rate (L1 Reads)
  - 92%
- Global Load L2 Transactions/Access
  - 23.4
- Maximum Divergence
  - 65.7%
- The overhead of histogram sort is low (~1ms)

Thanks!

https://github.com/littlemine
Xinlei Wang, 王鑫磊



https://www.3dgep.com/cuda-thread-execution-model/



GPU Execution Model

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