Hashing in the Scientific World

Tumblin, R.[1,2], Robey, R.W.[1],
Ahrens, P.[1,3], Hartse, S.[1,4]

[1]Los Alamos National Laboratory, [2]University of Oregon,
[3]University of California, Berkeley, [4]Brown University

LA-UR-13-22598, Version 2

A Computational Physicist's Perspective

- From the 2013 Summer Computational Physics Workshop with collaboration from two CS Majors - - Peter Ahrens, UC Berkeley and Sara Hartse, Brown University

- Looking deep into Algorithms, especially parallel algorithms, scaling, and memory usage was a different perspective for me.

Why Hashing?

- Hashing is an O(n) algorithm compared to current O(n log n) and even some O(n^2) algorithms
- Hashing is intrinsically parallel – no comparison operations needed
- Hashing is simple in contrast to many comparison-based approaches
- Example: Ordering alphabetically?

Spatial Operations for Computational Meshes

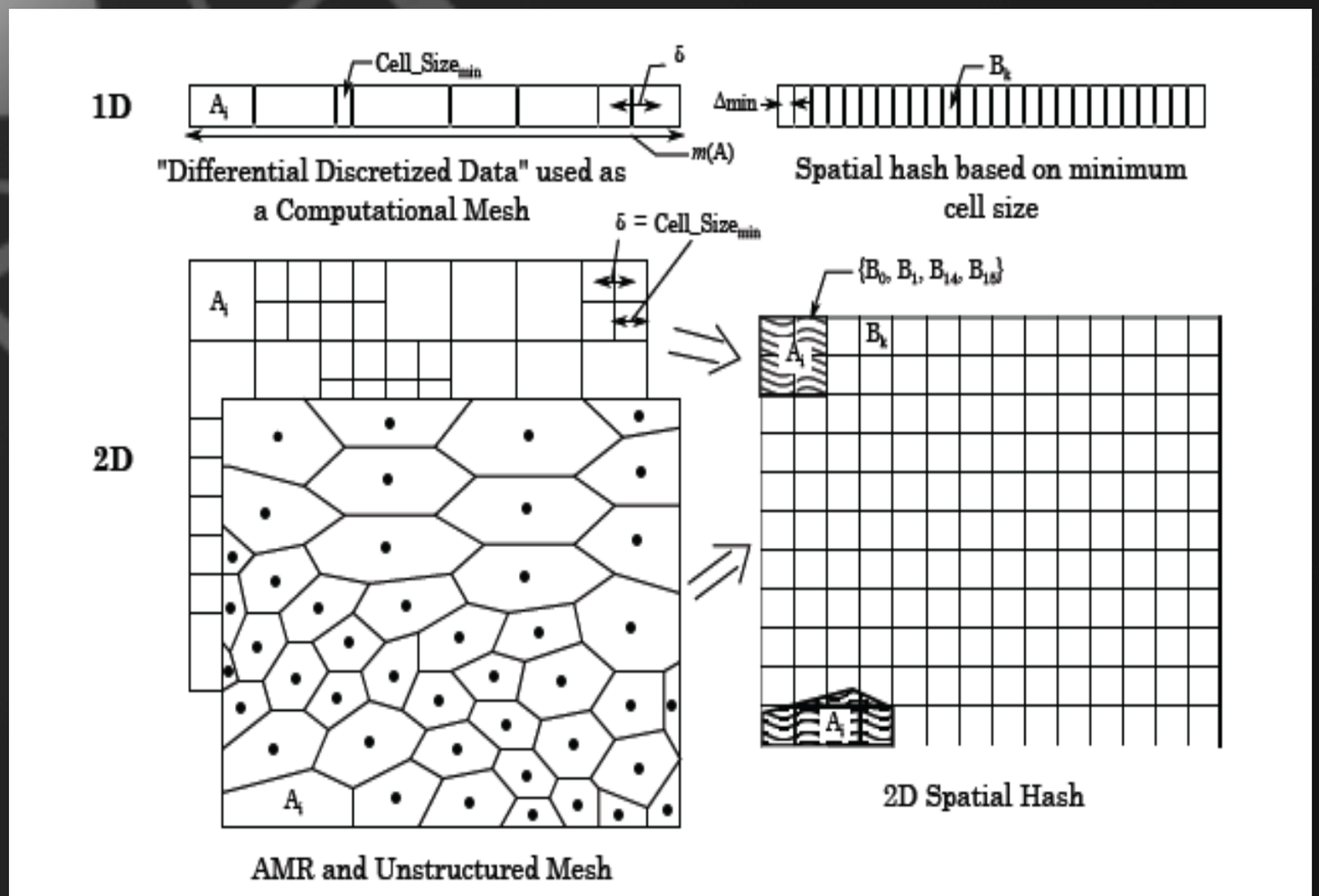
Types of Operations – any spatial

- Sorting
- Finding neighbors
- Remapping
- Table look-up

Hashing Techniques

- Perfect Hashing (prior work)
- Compact Hashing (our contribution)
- Binning or cell-lists using cell size set to interaction cut-off distance

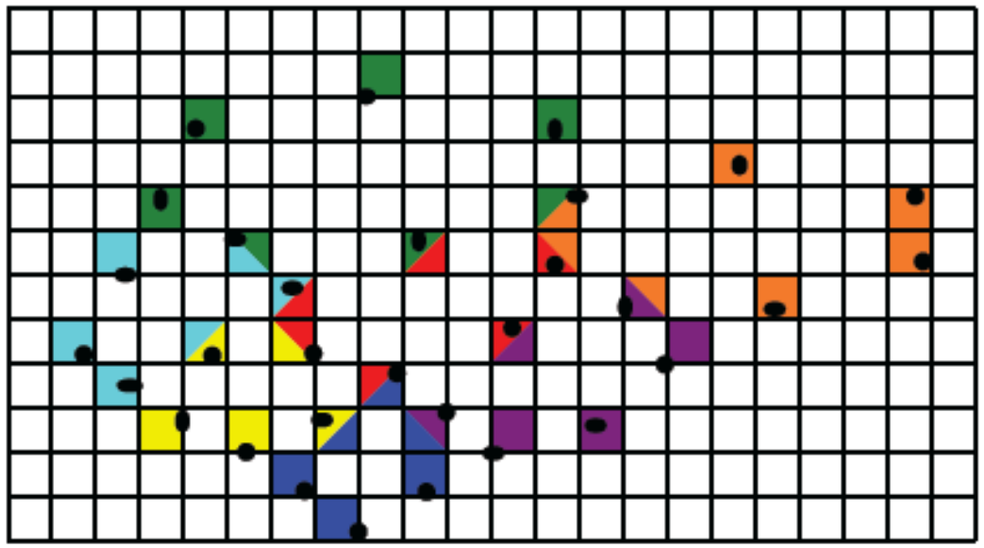
Perfect Hashing applied to Adaptive Mesh Refinement (AMR) and Unstructured Methods



- The key is to define a hash bin size small enough that only one spatial location will map to it.
- AMR – the hash size is set to the smallest cell size.
- Unstructured – based on minimum distances in cell

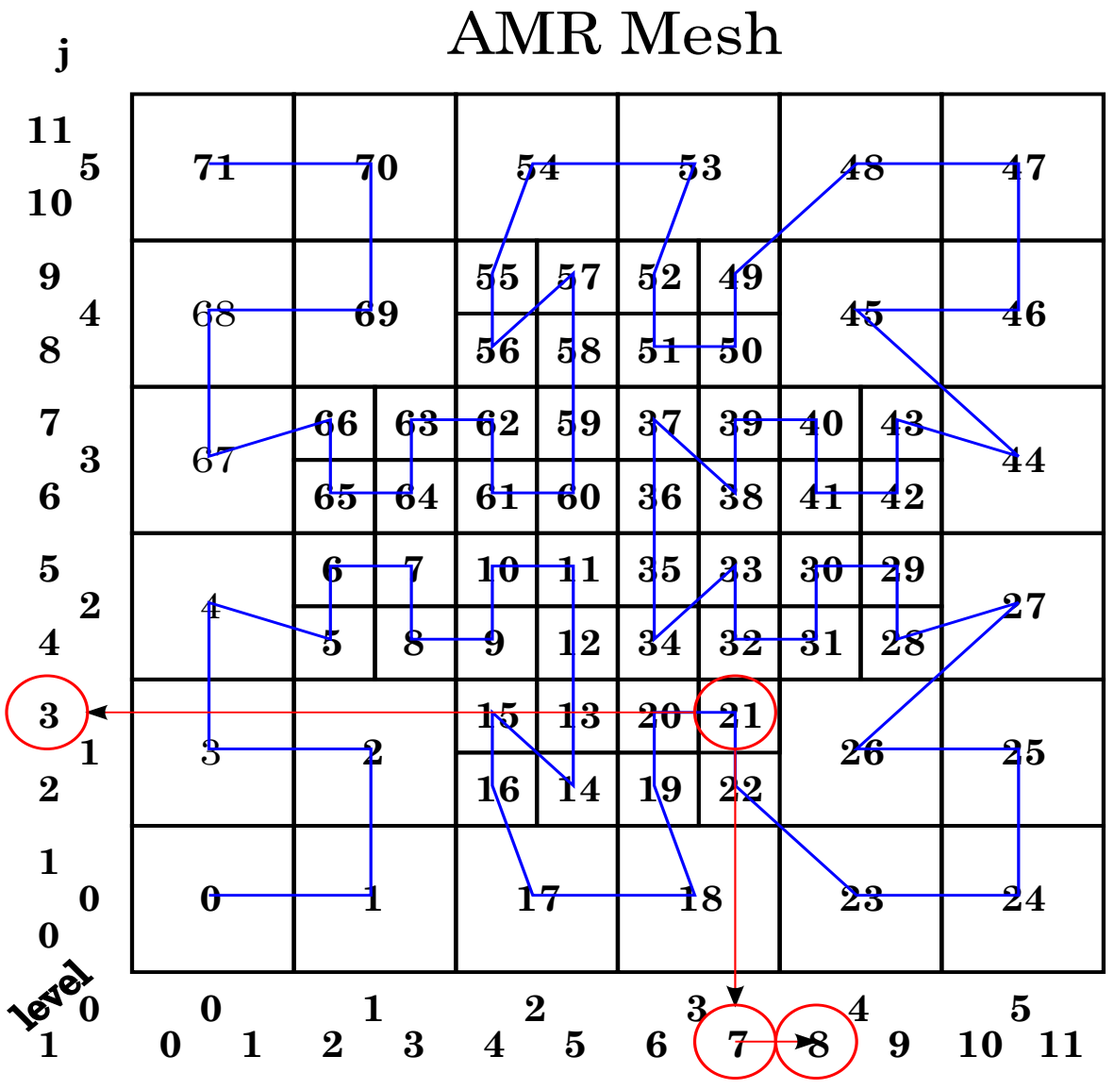
Robey, R.N., Nicholaeff, D., Robey, R.W., Hash-Based Algorithms for Discretized Data, SIAM Journal of Scientific Computing, Vol 35, Issue 4, C346-C368, July 2013

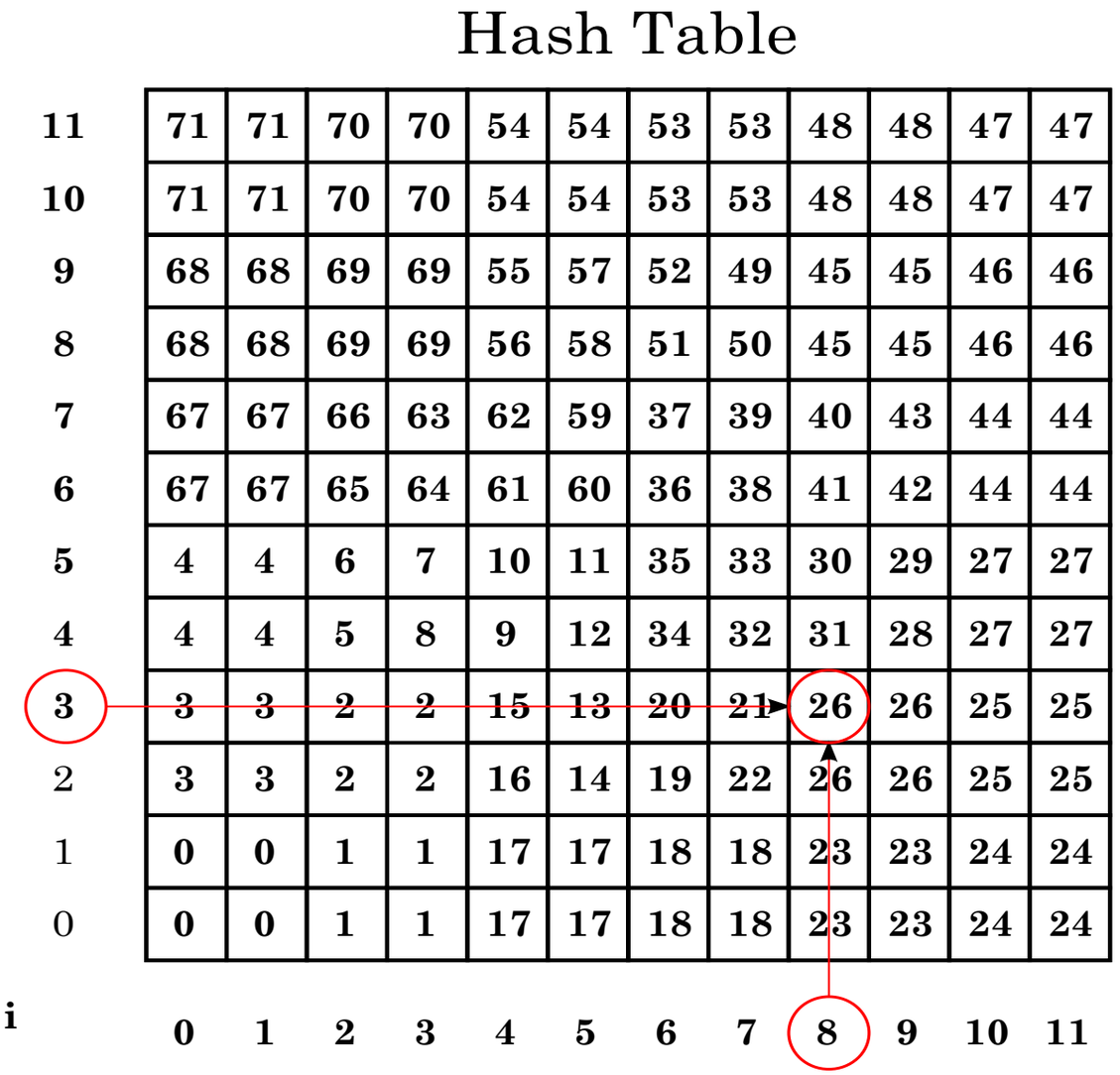
Unstructured Hash Concept



Nicholaeff, D. and Robey, R.N., Poster at 2012 LANL Student Symposium

Find neighbors by hash look-up





Each cell writes its cell number to hash buckets it covers

Right neighbor of cell 21 is at col 8, row 3. Look up in hash and it is cell 26

Speed-Up Summary

Note: Based on problem sizes (# of elements or cells) of around 2 million.
Reference CPU is generally accepted method for that operation: quicksort, kD-tree, and bisection.

	CPU Hash	NVIDIA	ATI	NVIDIA	ATI
Relative to	Reference CPU	CPU Hash		Reference CPU	
Sort	4.16	21.5	28.6	89.3	118.9
Sort 2-D	16.2	26.2	37.8	424.1	611.5
Neighbor	54.4	16.6	24.2	903.5	1316.0
Neighbor 2-D	75.5	19.1	19.1	1444.0	1445.3
Remap	18.4	26.9	48.1	495.2	885.8
Remap 2-D	13.6	42.2	61.6	574.0	837.8
Table	2.44	55.7	27.2	136.2	66.5

Speed-ups are a combined result of:

- replacing an O(n log n) algorithm with an O(n) algorithm
- harnessing the massively parallel compute capability of the GPU

Compact Hashing

Limits to Perfect Hashing:

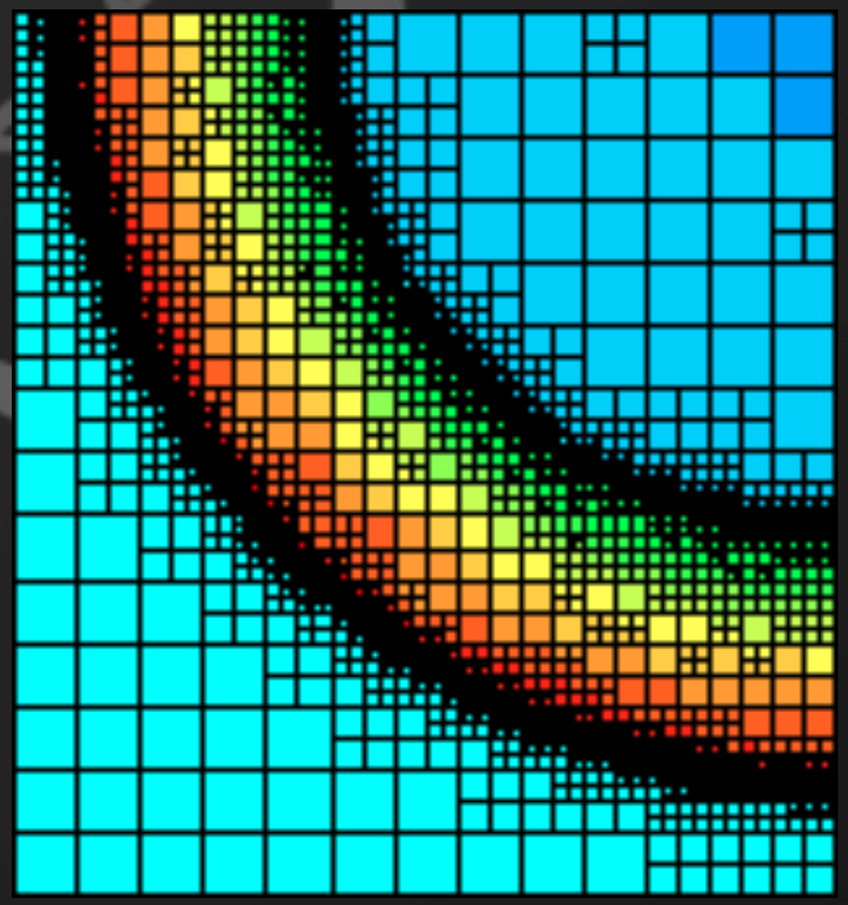
- Need to accurately determine minimum size to avoid collisions
- Memory requirements can grow as max to min cell size grows

Benefits of Compact Hashing:

- Compact hashing allows collisions
- Reduces memory requirements
- Scales to large problem sets

Let's look at the application of compact hashing to neighbor determination

Neighbor Determination For Cell-based Adaptive Mesh Refinement



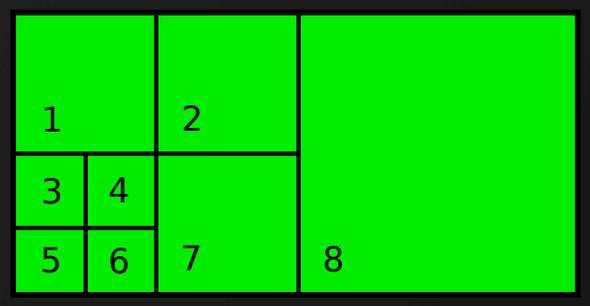
Δ_{min} = size of fine mesh grid elements

Compact Hashing exploits refinement rules of AMR

- We only allow one symmetric bisection between neighboring cells

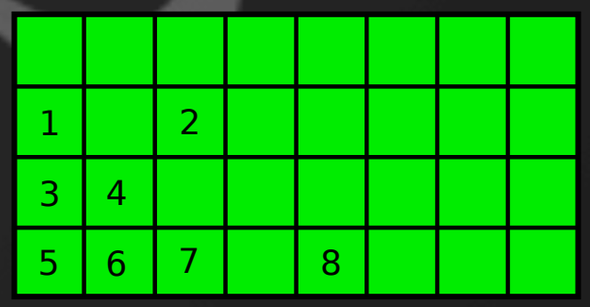
Compact Hash Algorithms for Computational Meshes

Adaptive mesh structure = 8 cells



Δmin

Compact



Sparse dataset

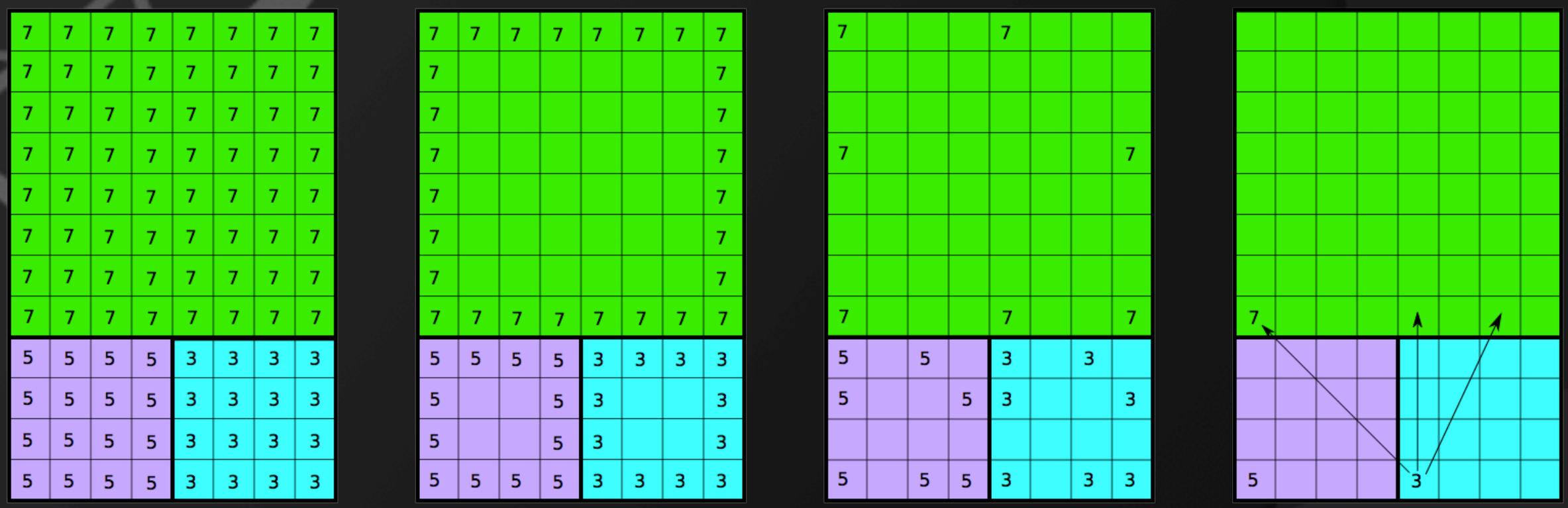
Perfect

1	1	2	2	8	8	8	8
1	1	2	2	8	8	8	8
3	4	7	7	8	8	8	8
5	6	7	7	8	8	8	8

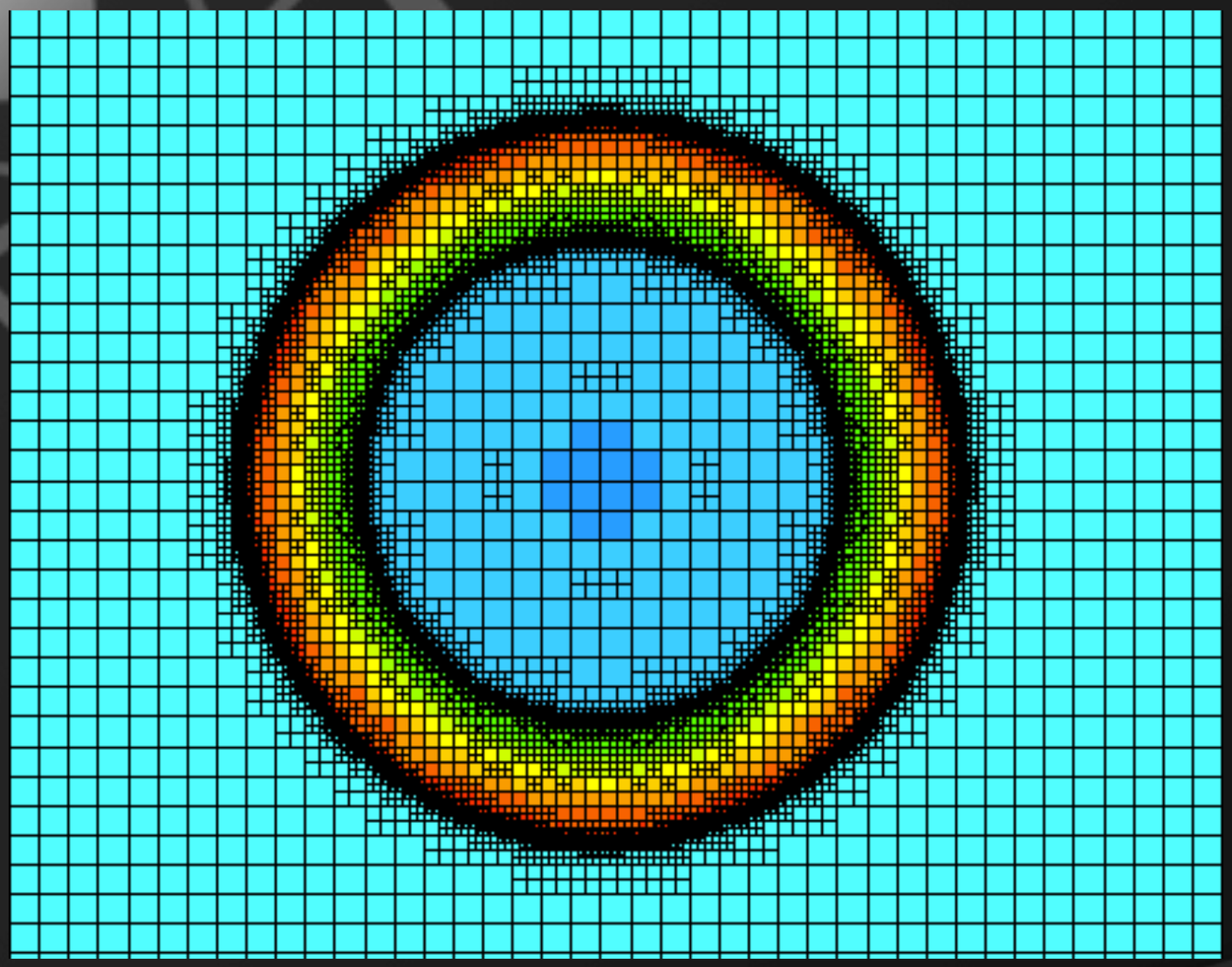
Fine mesh dataset

Can we find a way to reduce the number of hash bins yet still retain information about neighbors?

Taking Advantage of Data Structure: Memory Optimizations



Application to cell-based AMR shallow water hydrodynamics scheme

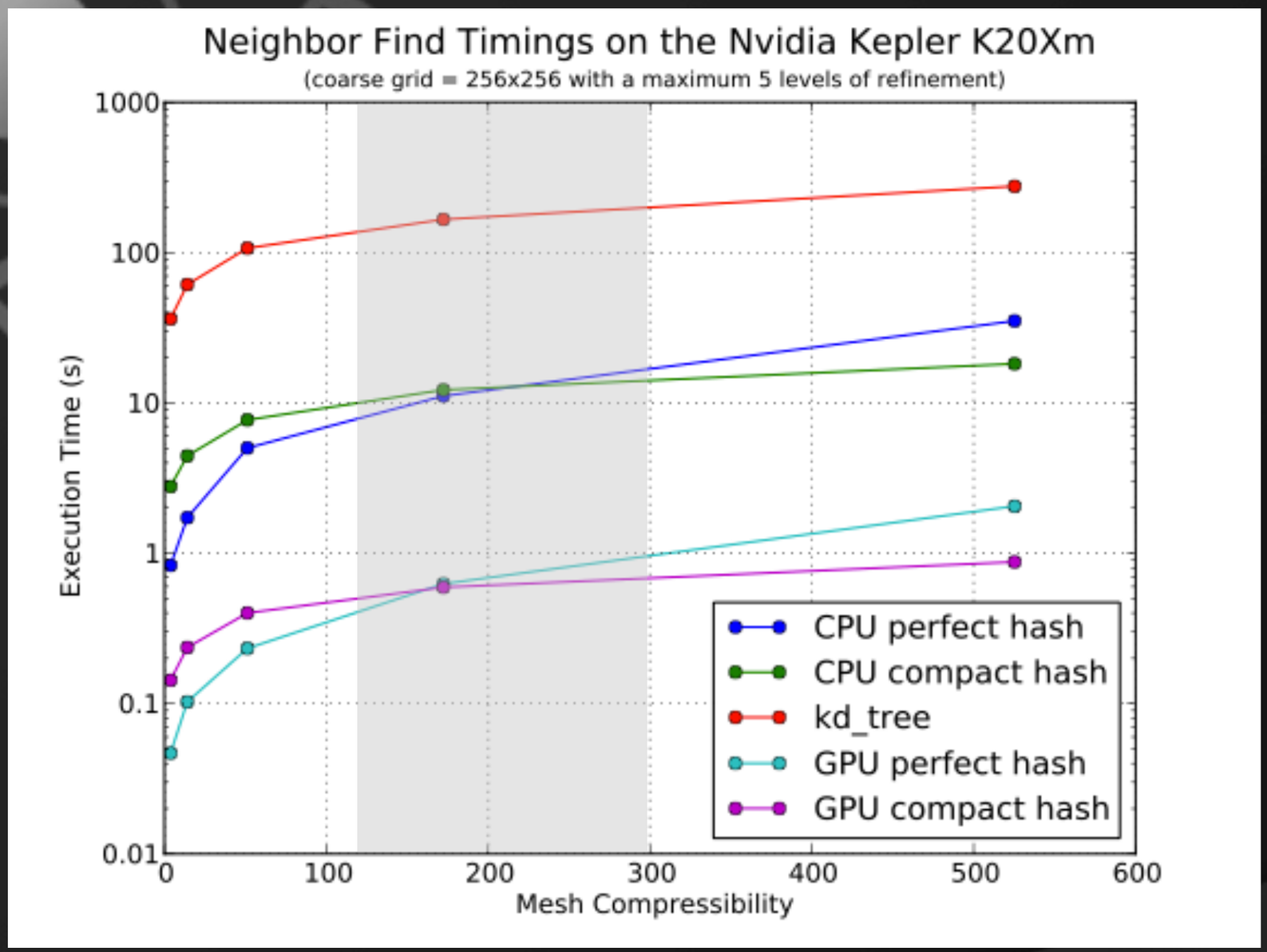


colors indicate height: red is higher

- Circular dam-break problem
- Developed for testing algorithms
- CPU and GPU capable using the OpenCL 1.1 standard

Performance Results:

Beyond 4 levels of refinement, the compact hash is faster and uses less memory



- GPU gives ~ 200X speed-up from binary tree method
- CPU gives ~20X speed-up from binary tree method
- GPU gives ~ 15X speed-up from CPU
- Compact hash scales to large datasets (slope)

← more collisions : more empty space →

Acknowledgments:

Los Alamos National Laboratory
Robert Robey
Scott Runnels
Zach Jibben
Sara Hartse
Peter Ahrens

UNIVERSITY OF OREGON

Los Alamos NATIONAL LABORATORY EST. 1943

References:

[1] Dan A. Alcantara. Efficient Hash Tables on the GPU. PhD thesis, UC Davis, 2011.

[2] Dan A. Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D. Owens, and Nina Amenta. Real-time parallel hashing on the gpu. ACM Trans. Graph., 28(5):154:1–154:9, December 2009.

[3] Rachel N Robey, David Nicholaeff, and Robert W Robey. Hash-based algorithms for discretized data. SIAM Journal on Scientific Computing, 35(4):C346–C368, 2013.

Collision Handling

- Compression function reduces the number of collisions by randomizing the data (See Carter and Wegman, 1979)
- Open Addressing handles collisions by searching for empty bin to insert data into

J. L. CARTER AND M. N. WEGMAN, Universal classes of hash functions, Journal of Computer and System Sciences, 18 (1979), pp. 143–154.

From Perfect to Compact Hashes