Hashing Meshes in Small Places

Sara Hartse[†], Rebecka Tumblin [§], Peter Ahrens [*]
[†]*Brown University* [§]*University of Oregon* [*]*University of California, Berkley*

LA-UR-13-25315

Abstract

We examine the potential of using hash-based algorithms to improve spatial operations required for unstructured meshes on Graphics Processing Unit (GPU) architectures. We investigate the performance of hash-based methods on Central Processing Units (CPU's) relative to a typical binary tree method and compare the speed of hashing on the GPU to the CPU. We implement memory optimizations which exploit the nature of the mesh allowing for compression of the data structure. The hashing method is then extended from a perfect hash to a compact hash using open-addressing. These implementations are tested across a variety of GPU architectures on both a randomly generated sample mesh and on an existing cell-based Adaptive Mesh Refinement (AMR) shallow-water hydrodynamics scheme.

Background

– Advances in GPU accelerated scientific computing necessitates the adaptation of serial algorithms to parallel architectures.

– Neighbor determination, a typically serial spatial operation in AMR computing methods, can act as a limiting factor in performance and execution times.

– Hashing is an intrinsically parallel, non-comparison based search data structure whereby a given floating point value passed to a hash function is mapped to an integer that fully describes its location within a hash table.

– Robey et al. demonstrated that meshes can be treated as discretized data allowing for a hash-based approach to be implemented for spatial operations such as neighbor finding, sorting or remapping.

– Our most basic hash method maps the mesh's data keys to a hash table corresponding to the spatial layout of the grid at the finest level of refinement.

– In our sample problem, we explore the benefits of implementing a compact versus a perfect hash in a random mesh of cells at varying levels of refinement. The mesh is smoothed such that there is no more than one level difference between adjacent cells.

Hashing

– A perfect hash ensures a location for each value and therefore requires the creation of a table as large as the largest possible hash code.

– A compact hash compresses the hash codes into a more managable range and requires a method of resolving the resulting collisions (multiple values being written to the same bucket).

– Open Addressing is a method that deals with collisions by skipping over filled buckets based on a probe sequence (linear or otherwise) until an empty one is reached.

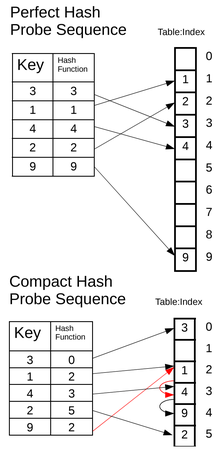


Figure 1: Hashing Methods

Algorithms

Timings of k-D Tree method relative to Perfect Hash (seconds)

Number of cells	k-D Tree (CPU)	Hash (CPU)
222739	1.791057	0.011638
383059	3.151747	0.032319
589900	5.322889	0.097610
804757	7.454789	0.259242
1005373	9.517877	0.884882

– The k-D tree method, a binary search method that can build branches based on spatial coordinates, is much slower than the perfect hash method for this size of mesh. However, as the highest level of refinement increases (which results in more, smaller cells) the gains from the hash method begin to decrease.

– Apart from speed-up on the CPU, the hash method has the advantage of consisting of highly parallel operations which makes it much simpler to transfer to the GPU. This was done using OpenCL.

Perfect Hash on the GPU

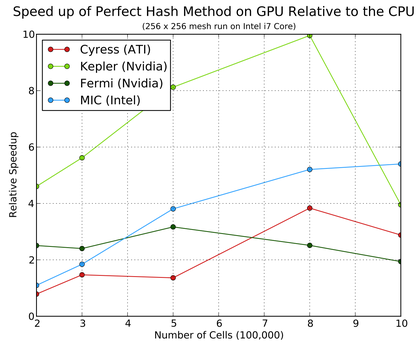


Figure 2: The parallel hash was tested on Nvidia's Tesla K20 (Kepler), Nvidia's Tesla M2090 (Fermi), ATI's Firepro V7800 (Cypress) and Intel's Many Integrated Core (MIC) and all devices show speed up from the CPU.

Memory Optimizations

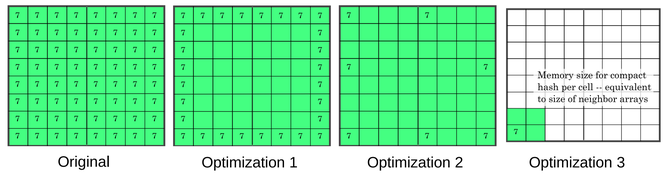


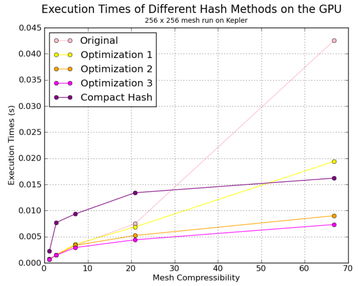
Figure 3: The original method and three levels of optimization for neighbor finding.

– Optimization 1 only writes to the outermost cells, 2 only to seven outer cells and 3 reduces it to only one write with three reads, each checking for a finer, same size, or coarser cell.

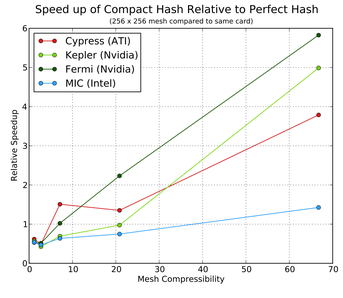
– These optimizations speed up the execution time of the perfect hash. Additionally, as the number of necessary buckets is reduced, the overall memory usage is compressed

– We define mesh compressibility as the ratio of the maximum number of cells on the finest level of the mesh to the maximum number of cells achieved in the simulation.

Compact Hash



Each optimization adds improvement from the original perfect hash, but the compact hash is still slower until the compressibility reaches a certain point.



The compact hash has reduced setup costs relative to the perfect hash, but the Open Addressing write is more expensive. After a certain compressibility is reached, each compact hash shows speed-up from its perfect hash counterpart.

Applications

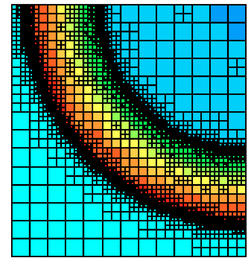
CLAMR

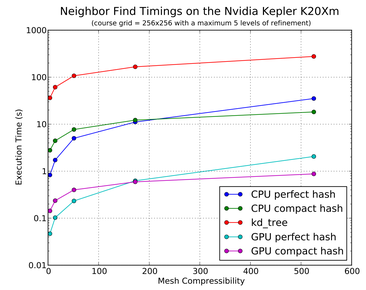
– CLAMR is a second-order accurate shallow water hydrodynamics scheme evolved on a cell-based Cartesian adaptive mesh which makes use of general process GPUs using the OpenCL 1.1 standard.

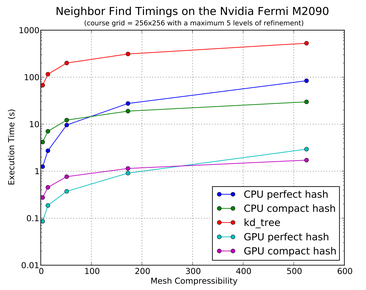
– CLAMR is built on a heterogeneous platform utilizing GPU parallelization in tandem with CPU memory management to achieve significant speed-up and performance.

– CLAMR employs hashing methods to replace the serial algorithms used in neighbor determination on the adaptive mesh.

– Here we explore the performance of these hashing methods across different devices within the context of a physical application.



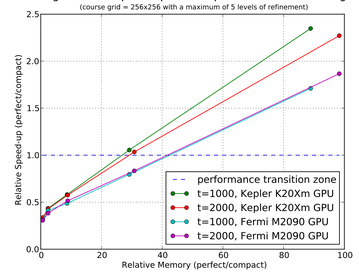




– The perfect hash achieves higher computational speed at low compressibility but eventually the compact hash has increased speed and reduced memory usage.

– This occurs because the compact hash requires extra operations for querying and accessing keys due to data collisions while the perfect hash has higher memory cost for initialization sparse hash tables.

– Due to the symmetry of the spherical wave, a longer iteration corresponds to a larger wave radius, and therefore, the mesh will adapt a larger number of refined cells to capture the steep gradients around the wavefront.



Conclusion

– The compact hash method is an extension of the perfect hash, offering significant speed-up and memory savings, especially with larger meshes.

– Meshes can be treated as discretized data allowing for hash-based approaches to be implemented for spatial mesh operations and hashing functions provide a new and innovative approach for adapting serial algorithms to parallel architectures.

– Utilizing and optimizing GPU parallelization and heterogeneous platforms in physics applications is numerically feasible on unstructured meshes, and offers new avenues for the future of computational sciences as we approach the age of exascale high performance computing,

For a detailed explanation of hashing for spatial operations see:
Robey et al. *Hash-based algorithms for discretized data.* SIAM Journal of Scientific Computing, LA-UR 12-01566, 2012.
For more information on parallel hashing methods see:
Alcantara. *Effcient Hash Tables on the GPU.* Diss. University of California Davis, 2011.