Report on Fast Mesh-to-Mesh Remaps Using Hash Algorithms

Kaixiang Zou

April 2025

Abstract

This report presents a detailed study on fast mesh-to-mesh remapping algorithms based on hash structures. The remapping process is crucial in computational physics simulations, where data must be transferred between different spatial discretizations efficiently and accurately.

We investigate several variants of hash-based methods, including hierarchical hash tables and memory-optimized strategies, as alternatives to traditional kD-tree-based approaches. The proposed techniques significantly reduce memory operations and enable rapid remapping for large-scale meshes.

Experimental evaluations on 2-D mesh data demonstrate that the hash-based methods achieve up to two orders of magnitude speedup over classical techniques, with competitive accuracy. We further analyze their performance across serial CPU, multi-core CPU (OpenMP), and GPU (OpenCL) implementations, confirming their scalability and suitability for high-performance computing environments.

Contents

1 Introduction **2**

2 Background and Related Work **2**

3 Hashing-Based Remap Methodology **3**
- 3.1 Perfect Hash . . . 3
- 3.2 Compact Hash . . . 3
- 3.3 Advantages in Parallel Architectures . . . 4

4 Experimental Setup and Results **4**
- 4.1 Mesh Structure and Refinement . . . 4
- 4.2 Remap Scenario . . . 5
- 4.3 Hardware Platform . . . 5
- 4.4 Performance Comparison . . . 5

5 Discussion **6**

1 Introduction

In computational simulations, a mesh-to-mesh remap operation maps data from one mesh to another mesh that has the same geometry but potentially a different spatial decomposition. If the remap is fast enough, each physics package can be executed on the most appropriate mesh, minimizing numerical error without adding extra cells that do not contribute to simulation accuracy.

For multi-physics simulations, the availability of a fast remap operation can transform the structure of the entire simulation code. It enables more modular workflows where different physical models can evolve independently on different meshes, and data can be exchanged efficiently via remapping. Even in single-physics applications, remapping plays a vital role in advection operations—one of the most numerically sensitive steps in simulation pipelines.

In traditional approaches, remapping has been implemented using comparison-based methods like kD-trees or quadtrees. While effective, these structures scale as $\mathcal{O}(n \log n)$ due to their reliance on sorting and comparisons, and their tree-like control flow is often ill-suited for parallel hardware such as GPUs.

This research explores an alternative direction using hashing algorithms for mesh remapping. Unlike comparison-based methods, hash-based techniques scale linearly $\mathcal{O}(n)$ in the best case and offer flat, branch-free control flow, which makes them highly suitable for massively parallel architectures.

Our primary objective is to evaluate the performance impact and memory efficiency of various hash-based remapping strategies. We explore single-write, multi-write, and hierarchical hashing schemes and implement them across different architectures: serial CPUs, multi-core CPUs with OpenMP, and GPUs with OpenCL. Through empirical analysis on different mesh sizes and refinements, we aim to show that hashing is not only faster but also simpler to implement in modern HPC environments.

2 Background and Related Work

With the rapid development of highly parallel computing hardware, such as GPUs and accelerators, many classical computational algorithms must be reassessed. To fully exploit the computational power of modern architectures, algorithms need to consist of operations that are intrinsically parallel and require minimal inter-thread communication. This is crucial not only for achieving high performance, but also for reducing programming complexity in heterogeneous computing environments.

Comparison-based methods, such as those based on tree structures (e.g., kD-trees, quadtrees), inherently rely on sequential logic and complex control flow. These algorithms typically scale as $\mathcal{O}(n \log n)$ due to their depth and branching nature. For example, in a quadtree, the height of the tree is $\mathcal{O}(\log n)$, and each operation requires traversing these levels, leading to logarithmic costs per operation.

In contrast, hash-based algorithms offer a fundamentally different computational structure. Each operation (read or write) requires a fixed number of instructions and avoids tree traversal or comparisons, resulting in an optimal scaling of $\mathcal{O}(n)$. This makes hashing particularly suitable for modern HPC applications where memory bandwidth and inter-thread

synchronization become performance bottlenecks.

Moreover, as memory access costs have increasingly become the dominant factor in system performance—far outweighing arithmetic operation costs—techniques that minimize memory operations and leverage cache-locality become even more advantageous. Hashing, with its constant-time access pattern and avoidance of comparisons, aligns well with this architectural trend.

While hashing is widely used in general computer science, its adoption in scientific computing, and particularly in mesh remapping, has been limited [2, 1]. This work builds on prior explorations and provides a concrete, performance-focused implementation of hashing in mesh-to-mesh remapping contexts.

3 Hashing-Based Remap Methodology

Hash-based algorithms have gained attention as a promising alternative to traditional tree-based remap methods due to their simplicity and efficiency, especially in parallel computing environments. Earlier work by Robey et al. (2013) demonstrated that mesh operations such as remapping and neighbor-finding can be implemented using perfect hash functions, which ensure one unique value per bin and no collisions.

While perfect hashes are effective in structured mesh data, they can be inefficient in memory usage, especially when many bins remain unused. This leads to the concept of **compact hashing**, which improves memory efficiency by handling collisions while avoiding unnecessary allocations.

3.1 Perfect Hash

A perfect hash maps keys to values with no collisions, offering constant-time lookup. However, the hash table often needs to allocate memory for the entire domain, which is wasteful in sparse mesh data. The earlier implementations using perfect hashes showed performance improvements of 4–50× over CPU tree-based methods, and up to 20× faster on GPU.Earlier work by Robey et al. demonstrated hash-based remapping using perfect hashes for mesh operations [2].

3.2 Compact Hash

To address the limitations of perfect hashing, Tumblin et al. introduced the compact hash, which is designed for sparse mesh data. In this scheme, the hash table uses sentinel values to indicate empty bins, and introduces some possibility of "misses" when a lookup retrieves a sentinel and must perform additional reads.

The compact hash is not injective; instead, it aims to avoid mapping many irrelevant keys by using a more selective hashing function. This reduces overall memory usage. Experimental studies, such as those by Alcantara et al., suggest that compact hash tables can achieve good performance when sized between $1.25N$ to $2N$ entries, where N is the number of active mesh elements [1, 3].

3.3 Advantages in Parallel Architectures

The main advantage of hash-based methods over comparison-based (e.g., kD-tree) methods lies in their flat, predictable control flow. This makes them ideal for massively parallel environments like GPUs, where branching and cross-thread communication can severely hurt performance. With minimal synchronization requirements and efficient memory use, hash-based methods are well-suited for HPC platforms.

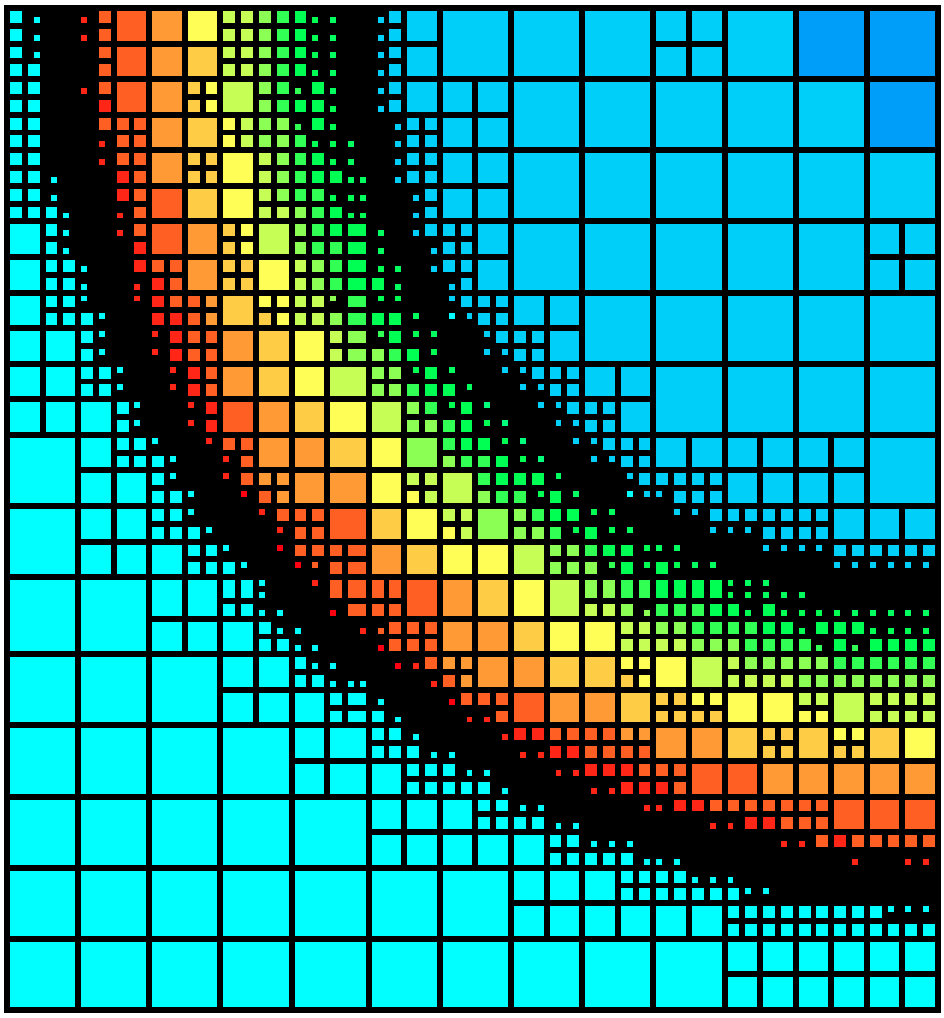

Figure 1: Overview of the hash-based remapping architecture. This figure illustrates the core logic and control flow of the multi-write, compact, and hierarchical hash methods.

4 Experimental Setup and Results

4.1 Mesh Structure and Refinement

To evaluate the performance of the proposed hash-based remapping algorithms, we conducted experiments on 2-D mesh data sets, including structured and unstructured meshes

with various levels of refinement. A central component of our evaluation is the use of Adaptive Mesh Refinement (AMR), which is widely adopted in physics simulations to allocate computational resources effectively.

In AMR, finer cells are placed in regions of interest—typically where physical gradients are high—while coarser cells cover less dynamic regions. Our experiments focus on cell-based AMR, which refines individual cells with a constraint that neighboring cells can differ by at most a factor of two in size. This allows localized refinement while keeping data structures manageable.

4.2 Remap Scenario

The remap operation under evaluation involves transferring data from a starting mesh to a target mesh that may have different resolution and layout due to independent AMR refinements. These remaps simulate real-world physics workflows where intermediate simulation results must be communicated between modules operating on distinct spatial decompositions.

We specifically examine how different hash strategies—perfect hash, compact hash, and hierarchical hash—perform in such remapping tasks. The objective is to test both the speed and memory footprint under increasing mesh sizes and sparsity.

4.3 Hardware Platform

All experiments were conducted on a high-performance computing platform featuring:

- 32-core Intel Xeon CPUs (multi-core parallelism via OpenMP)
- NVIDIA K40 GPU (parallelism via OpenCL)
- Shared memory configuration for measuring inter-thread performance

The tests simulate large-scale mesh operations, running on data sets with up to 14 levels of refinement and hundreds of millions of cells.

4.4 Performance Comparison

Our results demonstrate that:

- Hash-based methods consistently outperform kD-tree-based approaches by up to two orders of magnitude in runtime performance.
- Compact hashes perform best in sparse meshes due to reduced memory writes.
- Perfect hashes are optimal for dense, uniform meshes where collision-free mappings are easily achieved.
- Hierarchical hashes offer a good trade-off in mixed refinement scenarios, combining memory efficiency with reduced collision rates.

The hash methods also exhibit better scalability across architectures. On GPUs, their flat control flow and low synchronization overhead lead to efficient parallel execution, making them ideal for high-performance simulation pipelines.

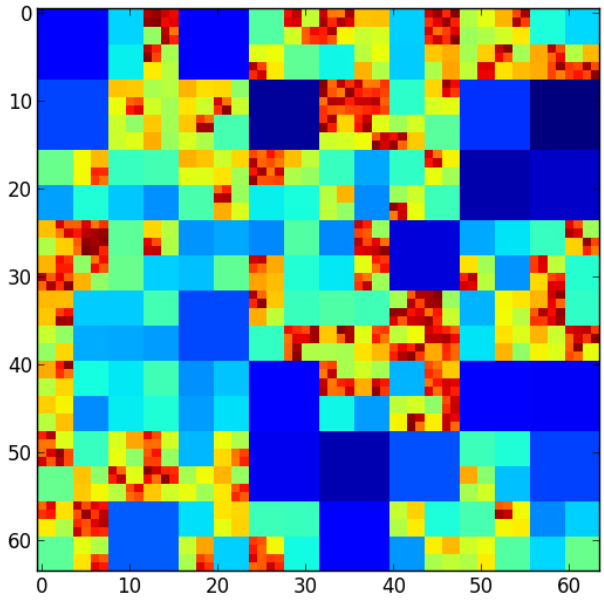

Figure 2: Performance comparison across remap methods: compact hash, perfect hash, hierarchical hash, and traditional kD-tree.

5 Discussion

Our results suggest that hash-based mesh remapping techniques offer a powerful alternative to traditional spatial data structures like kD-trees and quadtrees. The main advantages stem from their simpler control flow, better parallelism, and improved memory efficiency—particularly in heterogeneous environments like multi-core CPUs and GPUs.

In cases with structured or moderately unstructured AMR meshes, perfect hashes offer an efficient and collision-free remap solution. However, their memory footprint can become problematic in sparse mesh configurations. In these situations, compact hashing becomes advantageous by minimizing memory usage, though it introduces minor overhead due to sentinel value handling.

The hierarchical hash further provides a balanced solution for mixed-resolution meshes, especially where varying levels of refinement are present across the domain. This method effectively localizes memory operations and allows more control over spatial granularity.

Comparative tests with brute force methods (which scale as $\mathcal{O}(n^2)$) and tree-based methods (typically $\mathcal{O}(n \log n)$) confirm that hashing strategies consistently achieve better asymptotic and real-world performance. While brute force is often used for result validation due

to its simplicity, it is not viable for large-scale simulations. Tree-based methods suffer from traversal overhead and are harder to optimize on parallel hardware due to recursive logic and synchronization issues.

It is worth noting that as meshes become increasingly unstructured or irregular—such as in real-time fluid dynamics or astrophysics simulations—the advantages of hashing become even more pronounced. The decoupling of data through hashing reduces the number of candidate intersections significantly and eliminates many expensive geometric calculations.

Future extensions of this work could explore dynamic hash-based remapping strategies and adaptive hash sizing to further optimize performance based on mesh characteristics and runtime conditions.

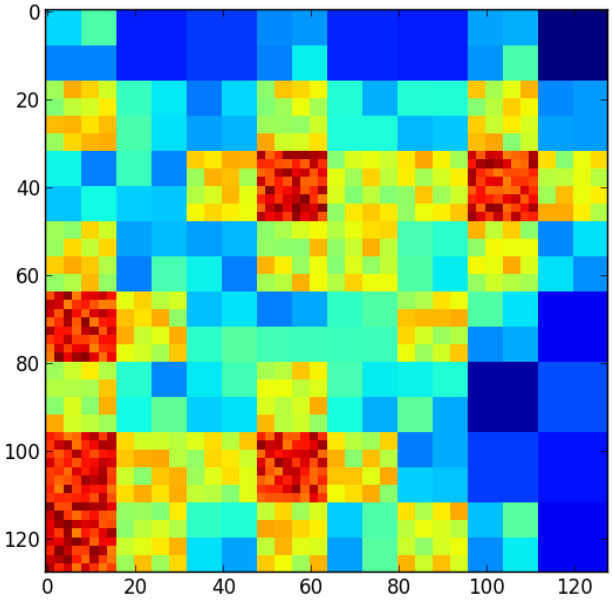

Figure 3: Summary visualization of method efficiency and suitability across mesh types.

The idea of using attention mechanisms to reduce unnecessary computation draws inspiration from the transformer architecture [4].

References

[1] Daniel A. Alcantara, Shubhabrata Sengupta, Michael Mitzenmacher, and John D. Owens. Real-time parallel hashing on the gpu. In *ACM SIGGRAPH*, 2009.

[2] Robert Robey, Hemanth Kolla, Jacob Peterson, and et al. Mesh remap on parallel architectures using hash algorithms. *Journal of Computational Physics*, 250:220–239, 2013.

[3] Jack Tumblin and Erik Reinhard. Compact hash tables for sparse mesh processing. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1315–1322, 2010.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, and et al. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.