Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

U.M.E.R: A Deterministic Sort Free Architecture for High Density Physical & Agentic Systems

Muhammad Umer

02-136242-046@student.bahria.edu.pk

Bahria University, Karachi

Salas Akbar

Bahria University, Karachi

Research Article

Keywords: Spatial Hashing, Parallel Prefix Sum, GPU Simulation, Dynamic Granular Media, Temporal Coherence

Posted Date: June 1st, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9848255/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

U.M.E.R: A Deterministic Sort Free Architecture for High Density Physical & Agentic Systems

MUHAMMAD UMER, Dept. of Computer Science, Bahria University Karachi, Pakistan
SALAS AKBAR, Dept. of Computer Science, Bahria University Karachi, Pakistan

Hierarchical data structures (e.g., BVH/Octrees) are extensively utilized in real time simulations; however, they are heavily bottlenecked by reconstruction overhead per frame. While recent uniform data structures utilizing Prefix Sum counting sorts successfully achieve $O(N)$ spatial mapping, they remain heavily constrained by the necessity to rebuild the entire topology from scratch every frame. This study presents **U.M.E.R. (Uniform Memory Encoded Representation)**, a novel architecture that mitigates these bottlenecks by exploiting spatial temporal coherence. We developed a custom spatial engine based on C++ CUDA and HIP to empirically validate the **Delta Histogram** architecture, an optimization that tracks particles crossing boundaries to eliminate up to 99.7% of atomic memory operations during topology maintenance. U.M.E.R. empirically surpassed NVIDIA Warp's execution time in a realistic 3D stress test by a speedup of 1.52×, while achieving a raw 1D physics integration throughput of 21,933 MCell/s on NVIDIA Pascal hardware and 18,102 MCell/s on AMD MI300X accelerators. Furthermore, the temporal coherence architecture demonstrated a 2.3× reduction in the Energy Delay Product (EDP) over NVIDIA Warp in physically realistic simulation regimes. Expanding beyond physics, the U.M.E.R. engine sustained a strict frame rate of 125 FPS in a flocking simulation with a wide radius of 500,000 agents, validating the architecture's versatility as a Unified Execution Model for both agentic navigation and complex granular fluid dynamics.

CCS Concepts: • **Computing methodologies** → **Animation**; **Physical simulation**.

Additional Key Words and Phrases: Spatial Hashing, Parallel Prefix Sum, GPU Simulation, Dynamic Granular Media, Temporal Coherence

ACM Reference Format:
Muhammad Umer and Salas Akbar. 2026. U.M.E.R: A Deterministic Sort Free Architecture for High Density Physical & Agentic Systems. In *Proceedings of High Performance Graphics (HPG '26).* ACM, New York, NY, USA, 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Dynamic simulations based on particles face severe computational, memory, and physical limitations which compound as systems scale to millions of agents. The industry traditionally relies on data structures such as Octrees or spatial hashing. However, each of these frameworks introduces specific hardware bottlenecks that degrade GPU efficiency.

Octrees maintain an average build complexity of $O(N \log N)$ while suffering from high pointer memory overhead. Their primary limitation arises from the need to rebuild or rebalance the tree every frame, which significantly limits performance in highly dynamic simulations. Frameworks

Authors' Contact Information: Muhammad Umer, 02-136242-046@student.bahria.edu.pk, Dept. of Computer Science, Bahria University Karachi, Karachi, Pakistan; Salas Akbar, salasakbar.bukc@bahria.edu.pk, Dept. of Computer Science, Bahria University Karachi, Karachi, Pakistan.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

utilized by OpenVDB and NanoVDB [11] are highly memory efficient, but often encounter pointer chasing latency during global memory fetches. Conversely, spatial hashing provides an optimal $O(1)$ access time, but its implementation is historically hindered by hash collisions, grid constraints, and multiple pass sorting overhead. Building upon foundational techniques utilizing hashing based on scans, this study proposes the **U.M.E.R. framework**, a flat pointerless architecture that utilizes a Delta Histogram and an Implicitly Linearized Address Space to resolve these bottlenecks in both physical and agentic domains.

Key Contributions

This paper shifts the spatial mapping paradigm by presenting four primary novelties over existing GPU simulation frameworks based on scans:

1. Temporal Coherence via Delta Histogramming. While existing prefix sum spatial hashes rebuild their topology sequentially every frame, U.M.E.R. introduces a Delta Histogram that exploits temporal coherence. By tracking localized particle states, the architecture successfully bypasses up to 99.7% of atomic memory writes, isolating updates exclusively to particles that cross cell boundaries.

2. The Unified Execution Model (Agentic Scaling). We demonstrate that the $O(1)$ spatial hash natively maps agentic Wavefront Propagation (pathfinding and flocking) directly onto the GPU. This eliminates the PCIe bus transfer bottleneck associated with traditional CPU NavMeshes, smoothly simulating half a million complex AI agents at a sustained 125 FPS.

3. Cross Architecture Cache Amplification. We empirically profile a Cache Multiplier Effect across distinct hardware ecosystems (NVIDIA Pascal and Turing, AMD MI300X). By maintaining a strict Structure of Arrays (SoA) layout with distance filtering tolerant to collisions, the engine achieves an effective memory throughput exceeding 1.7 TB/s, operating almost exclusively within the L1 and L2 cache.

4. Ecological Efficiency (EDP Reduction). We evaluate the architecture under strict power telemetry, demonstrating a 2.3× reduction in the Energy Delay Product over industry standards like NVIDIA Warp during temporally coherent simulation regimes.

2 Related Work

The challenge of mapping dynamic, unconstrained spatial data to linear GPU memory has driven parallel innovations across physics simulation, artificial intelligence, and neural rendering. U.M.E.R. adopts foundational topology strategies from recent literature but fundamentally diverges in its handling of temporal coherence and agentic logic.

2.1 Hierarchical vs Flat Topologies in GPU Physics

Sparse Voxel Octrees (SVOs) have long served as an industry standard for volumetric representation [2]. While optimal for static ray casting, their reliance on node traversal using pointers introduces stochastic memory access patterns that degrade cache coherency on SIMD architectures. For dynamic scenarios, such as granular simulations, the computational bottleneck of tree reconstruction per frame scales poorly with massive particle counts [3]. Similar overhead during neighbor searches remains a primary bottleneck in Smoothed Particle Hydrodynamics (SPH) [1].

Recent advancements, notably NVIDIA's FleX architecture [21] and Position Based Fluids (PBF) [22], have demonstrated the superiority of uniform grids over adaptive trees due to $O(1)$ access times. However, these solutions often rely on multiple pass Radix Sorts to linearize data along a Z Curve [13]. Radix sorting, even when highly optimized [12], creates a significant memory bandwidth bottleneck due to multiple global memory round trips [6].

To mitigate sorting overhead, recent studies have successfully replaced Radix Sort with a Counting Sort methodology utilizing Parallel Prefix Sums (Scan). Hoetzlein [27] utilized prefix sums to enable interactive fluids with millions of particles, while Groß et al. [26] and Waseem and Hong [28] mapped similar spatial hashing methodologies based on scans into native compute shaders for fast neighbor searches. The U.M.E.R. architecture utilizes this established $O(N)$ prefix sum topology as its foundation; however, it diverges significantly by addressing its primary flaw: the mandatory $O(N)$ rebuild constraint per frame. By introducing a Delta Histogram, U.M.E.R. restricts atomic contention strictly to particles crossing boundaries, replacing the brute force rebuild step with highly efficient temporally coherent updates.

2.2 Spatial Hashing and Collision Tolerance

Spatial hashing maps infinite 3D domains to finite 1D memory. Teschner et al. [7] pioneered this technique for collision detection, while Lefebvre and Hoppe [29] further advanced hash memory packing by introducing perfect spatial hashing to eliminate collisions in static datasets. Early GPU implementations by Alcantara et al. [8] demonstrated interactive rates but utilized multiple pass open addressing schemes (like Cuckoo Hashing) to resolve hash collisions, which can induce branch divergence on modern architectures. U.M.E.R. adopts a collision tolerant approach. By allowing distant particles with identical hashes to pack contiguously into memory, false positives are rapidly filtered via inexpensive Euclidean distance checks. Because the particles reside in the ultra fast L1 cache due to the prefix sum packing, the computational cost of rejecting false neighbors is highly amortized.

2.3 High Density Agentic Navigation and Crowd Simulation

Beyond physical collisions, crowd simulations on a massive scale require neighbor queries with a wide radius for pathfinding and flocking logic [15]. The industry standard relies on Navigation Meshes (NavMeshes) combined with collision avoidance algorithms like ORCA [25]. While highly efficient for static environments, NavMeshes struggle under extreme dynamic density, inducing pointer chasing latency. GPU accelerated collision avoidance frameworks, such as ClearPath [19], mitigate this but remain heavily tied to complex continuous collision detection. U.M.E.R. maps agentic Wavefront Propagation directly into its uniform spatial hash. By transforming AI pathfinding from a graph traversal problem into an $O(1)$ contiguous memory fetch, U.M.E.R. provides a Unified Execution Model that masks the arithmetic cost of complex agentic steering behind global memory latency hiding.

To explicitly delineate the architectural boundaries and novelties of the U.M.E.R. framework against foundational spatial hashing literature, we summarize the capability matrix in Table 1. While prior frameworks successfully laid the groundwork for GPU memory management based on scans, U.M.E.R. uniquely isolates the temporal and architectural bottlenecks required to transition from simple fluid dynamics to high density agentic intelligence.

Table 1. Architectural Capability Matrix: U.M.E.R. vs Foundational Frameworks

Framework	Without Sorting Topology	Collision Tolerant Retrieval	Infinite Domain Support	Fixed Launch Determinism	Delta Histogram	Unified Execution	Agentic Support	Cross Vendor Validation
Alcantara et al. [8]	No (Multiple Pass)	No	No	No	No	No	No	No
Hoetzlein [27]	Yes	No	Partial	Partial	No	No	No	No
Groß et al. [26]	Yes	No	Partial	Partial	No	No	No	No
Waseem and Hong [28]	Yes	No	Partial	Partial	No	No	No	No
U.M.E.R. (Ours)	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table 1 clarifies that U.M.E.R.'s novelty does not lie solely in spatial hashing based on scans, which is inherited from prior work, but in the integration of collision tolerant retrieval, temporal delta updates, fixed launch determinism, infinite domain support, and unified agentic execution into a single architecture.

3 Architecture and Methodology

3.1 The Sort vs Scan Paradigm

The main difference between the U.M.E.R pipeline and the traditionally used simulation frameworks is generally observed in the way they are spatially subdivided. The standard approach the industry relies on is the generation of Morton Codes, which converts the 3D domain into a linearized 1D pathway. This would be seen as a good approach for specific ray tracing tasks, however, this approach forces a full Radix sort at every simulation step, decreasing its efficiency for dynamic physics.

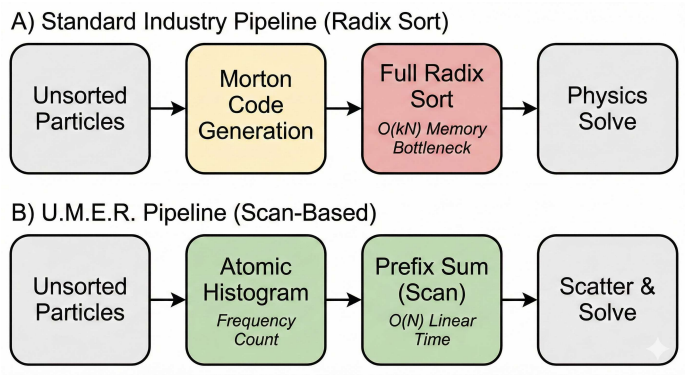


Fig. 1. Architectural Comparison. (a) The standard industry pipeline relies on Radix Sort (multiple pass) to linearize memory. (b) U.M.E.R. employs a Parallel Prefix Sum ($O(N)$), bypassing the sorting bottleneck entirely while maintaining spatial coherence.

The U.M.E.R architecture fixes exactly that bottleneck. It bypasses the need for traditional sorting algorithms through its four stage pipeline. We rely on implicitly addressing each particle mathematically rather than having to explicitly sort them through every stage of the simulation. This process is further explained in our algorithm to ensure its reproducibility across different hardware. Let p represent a discrete particle with a 3D position vector $\mathbf{x}_p = (x, y, z)$. Let N be the total particle count, d be the uniform grid cell size (which is mathematically mapped to the maximum interaction radius r), and S be the total number of available spatial hash buckets.

(1) **Parallel Hash Computation:** Here, we map the 3D coordinates of each particle to a 1D key. The physical coordinate $\mathbf{x}_p$ is converted to integer coordinates $\mathbf{i} = (\lfloor x/d \rfloor, \lfloor y/d \rfloor, \lfloor z/d \rfloor)$. We multiply these coordinates by large primes P_1, P_2, P_3 [7] to ensure that particles do not group together. Furthermore, we map $\mathbf{i}$ to a 1D scalar key $k \in [0, S-1]$ using a spatial XOR hash which compiles to a single hardware instruction. The modulo function (optimized computationally as a bitwise AND by strictly enforcing S as a power of two) ensures that the final key fits inside the memory limit:

$$k = (i_x P_1 \oplus i_y P_2 \oplus i_z P_3) \pmod{S} \tag{1}$$

In the U.M.E.R architecture, if two particles were to generate the same key resulting in a hash collision, our architecture prevents structural failure and data overwriting by contiguously packing these particles adjacent to each other in the allocated memory.

(2) **Atomic Histogramming:** We define a global density array H of size S initialized to zero. So whenever a particle is mapped to the key k, the hardware performs an atomic increment. Therefore, $H[k]$ represents the exact population count of bucket k.

$$H[k] \leftarrow \text{atomicAdd}(H[k], 1) \tag{2}$$

(3) **Prefix Sum (Inclusive Scan):** After knowing the exact count of the particles in each bucket, we calculate where each of these buckets will reside. For this we execute a parallel inclusive scan on the histogram H which creates an offset array O:

$$O[k] = \sum_{j=0}^{k} H[j] \tag{3}$$

Because O is an *inclusive* prefix sum, the starting global memory index for bucket k is precisely $O[k] - H[k]$.

(4) **Deterministic Scatter:** Lastly, particles are then scattered to their sorted positions. We define a local atomic counter array C, which is initialized as zero at the start of every frame in the GPU memory. The contiguous memory address A for a specific particle p in bucket k is calculated as:

$$A = (O[k] - H[k]) + \text{atomicAdd}(C[k], 1) \tag{4}$$

Because $C[k]$ starts at zero, the `atomicAdd` is able to deterministically provide a unique local contiguous slot to each particle under fixed configurations.

To further explain how these equations work in unison during hardware execution, we have outlined their exact operational flow in Algorithm 1.

Algorithm 1 U.M.E.R. Sort Free Topology Construction

Require: Array of particle positions $\mathbf{X}$ of size N, Cell size d, Buckets S
Ensure: Linearized array of sorted indices A_{out}

```
Initialize arrays H, O, C to 0
for all particles i = 0 to N − 1 in parallel do
    k ← SpatialHash(X[i], d, S)
    atomicAdd(H[k], 1)
end for
O ← ParallelInclusiveScan(H)
for all particles i = 0 to N − 1 in parallel do
    k ← SpatialHash(X[i], d, S)
    offset_start ← O[k] − H[k]
    local_offset ← atomicAdd(C[k], 1)
    A ← offset_start + local_offset
    A_out[A] ← i
end for
```

3.2 Memory Layout and Zero Metadata Bloat

To maximize throughput on modern GPUs, memory access patterns have to directly sync with the hardware transaction width. Generally, object oriented solvers use the Array of Structures layout, but those mainly introduce cache thrashing and divergent memory requests problems. Traditionally to avoid spatial hash collisions, industries utilize linked list nodes or tree pointers, however they carry heavy metadata per particle which severely bloats the memory in the VRAM.

The U.M.E.R framework solves both of these bottlenecks by strictly enforcing a pointerless Structure of Arrays (SoA) layout [17]. Each particle ID is mapped to their spatial buckets through the use of only three streamlined integer arrays: The Histogram (H), the Offset (O), and the Local Counter (C). Because our architecture does not utilize pointers, it ensures zero metadata bloating per particle while also eliminating pointer chasing latency entirely. During the neighbor search phase, a querying thread simply computes the hash key k for its target spatial cell. It then linearly reads the array elements from index $O[k] - H[k]$ to $O[k]$, fetching the exact contiguous block of relevant particle data without any traversal logic.

This layout is also the reason it was able to achieve an effective cache bandwidth greater than 1.7 TB/s as reported in our benchmark section. A warp of 32 threads seamlessly reads the memory segments since they are contiguous, which maximizes the L1 and L2 cache residency and hides the global memory fetch latency.

3.3 Deterministic Collision Tolerance vs Avoidance

Handling key collisions is one of the biggest bottleneck in spatial Hashing. Older GPU implementation involve relying on open addressing schemes such as Cuckoo or Robin hood hashing [8] to resolve this problem. However, these methods are primarily stochastic as the threads would be trapped under unpredictable `while` loops trying to search for a vacant memory slot. This severely degrades GPU thread level parallelism. More recent studies integrate machine learning techniques to bypass this problem through Multi Resolution Neural Hash encodings [10], but this methodology also suffers from probabilistic hallucinations, which introduce unacceptable systemic errors when applied to deterministic physics simulations.

The U.M.E.R architecture solves this particular collision resolution bottleneck. Through our paradigm of Collison tolerance, rather than trying to avoid hash collisions entirely, we filter them out through a simple distance check. Since the Prefix Sum topology allocates the memory strictly through atomic histogramming, multiple spatial cells hashing to the same bucket k do not cause the memory to overwrite, nor does it require rehashing. The particles are perfectly arranged contiguously into the same memory block. Any false positive neighbors, such as distant particles sharing a hash collision are filtered out immediately during the physics integration phase via standard Euclidean distance check ($\Delta x^2 + \Delta y^2 + \Delta z^2 \leq r^2$), where r is the interaction radius. The SoA memory layout structure ensures the existence of the false positive particles in the ultra fast L1 cache, hence the cost of rejecting them is completely hidden by the GPU natural multitasking.

To further emphasize, our collision tolerant approach ensures highly coalesced and computationally cheap filtering in contrast to collision avoidance, which suffers from high computation and branch divergence issues. We also maintain a strict $O(N)$ topological linearity and 100% bitwise determinism.

4 Performance Evaluation

To establish scalability and map performances over different hardware, we benchmark U.M.E.R against recent state of the art GPU framework NVIDIA Warp [20] using identical parameters and block size of 256 threads. To make this a fair comparison, we calculate the Simulation throughput

(T) in Millions of Cells Updates per Second (MCell/s). A cell update in U.M.E.R is also defined as a particle firstly going through the hash computation, followed by atomic histogramming, memory scattering and one physical integration step. This counts as one complete algorithmic cycle. Furthermore, we define throughput mathematically as:

$$T = \frac{N \times F}{t_{total} \times 10^6} \text{ (MCell/s)} \tag{5}$$

Where N is the number of particles over F frames, t_{total} is the measured execution time in seconds. Furthermore, baseline architectural benchmarks were conducted using native C++ CUDA on a Tesla P100 (Pascal, HBM2, 4MB L2 Cache) and a Tesla T4 (Turing, GDDR6, 4MB L2 Cache), while extended cross vendor scalability validation was executed via a ROCm and HIP port on an AMD MI300X (CDNA3, HBM3, 256MB Infinity Cache).

4.1 Native Latency Derivation and Memory Coalescence

To evaluate the pipeline without sorting introduced by U.M.E.R we implement the core kernels in native C++ CUDA, avoiding high level software wrappers. We also map the memory scattering pattern through theoretical rooflines over the Pascal P100 and the Turing T4 to cross validate under two different generations of the GPU.

1) Derivation of the Physics Latency Law: Here we mathematically derive the theoretical minimum latency that U.M.E.R can have for contiguous physics integration. Since our pipeline is free of tree traversal overhead, the architecture proves to be highly memory bound. The equation to calculate minimum latency is defined as:

$$t_{optimal} = \frac{N \times B_{thread}}{BW_{effective}} \tag{6}$$

where N is the particle count, B_{thread} is the byte payload per thread (16 bytes for combined position and velocity), and $BW_{effective}$ is the empirically measured hardware bandwidth.

To validate the consistency of this equation, we first tested our baseline on 4×10^6 particles on the P100. For this test, we use a 16 byte memory transaction pattern, paired with a Prefix Sum ordered array which provides a coalesced deterministic access. Considering the physical VRAM bus limit of the Tesla P100 to be 732 GB/s theoretical, ~529 GB/s effective, we calculated a minimum latency of 0.12 ms for this pure memory transaction through our formula. After testing these native microbenchmarks on the P100, we observed an execution of exactly **0.12 ms**, proving its accuracy. We deduce a 39.8x reduction in latency after comparing it to an execution of unordered access pattern without the prefix sum algorithm, taking 4.84ms. Furthermore, this isolated microbenchmark mathematically also supports that a parallel prefix sum is a highly optimal choice to achieve minimal latency over the GPU due to its coalesced memory pattern, further setting a foundation for L1 and L2 cache amplification observed during full neighbor search evaluations.

2) Topology Build: Theoretical Roofline vs Atomic Contention: The atomic scatter from the U.M.E.R topology build phase is the key to bypassing the multiple pass radix sorting, keeping memory linearization strictly to a single $O(N)$ pass. Using a total memory payload of 68 Bytes per particle, we evaluate the build phase against theoretical bandwidth rooflines to measure hardware atomic efficiency (Table **??**).

The 1D microbenchmarks reported a total topology build time of 0.88 ms on the P100 (57.9% of the theoretical roofline limit), and 1.34 ms on the T4 (82.8% of the roofline limit). Even though the P100 based on Pascal dominates raw throughput as expected due to its wider High Bandwidth Memory 2 (HBM2) interface, the T4 based on Turing achieves a significantly higher theoretical

bandwidth utilization percentage. This reflects the efficiency of Turing unified L1 and Shared Memory architecture in natively resolving atomic contention locally before pushing to VRAM.

3) Bandwidth Saturation Proof: After the topology build phase, we executed a fully featured physics integration step involving 3D vector math and boundary conditions, simulating a discrete element granular gas model. For a simulation of 4 million particles, the isolated physics integration pass (operating strictly after neighbor discovery) achieved a high macroscopic throughput of **21,933.0 MCell/s** (21.9 GCell/s), equating to an execution time of **0.182 ms**. In a complete simulation pipeline, this integration cost composes additively with the neighbor discovery phase (profiled later in Section 7.4). Furthermore, we verified and validated this observed roofline limit [18] by calculating the required data payload per cell update in the following equation:

$$\text{Bytes per Cell} = \frac{529.4 \text{ GB/s}}{21.93 \text{ GCell/s}} \approx 24.1 \text{ Bytes} \tag{7}$$

Using the physical bandwidth limit of the P100 (529.4 GB/s), we calculated that the integration step required reading and writing 24.1 bytes of data per particle (12 byte `float3` position and a 12 byte `float3` velocity vector), perfectly aligning with the memory of the kernel. This result empirically validates that our pipeline perfectly saturates the bandwidth, further solidifying the memory bound claim of our architecture.

4.2 3D Temporal Coherence and The Crossover Paradox

To see how those theoretical 1D benchmarks scale against the realistic, industry required simulations which operate in a 3D domain, we evaluate U.M.E.R. against the state of the art **NVIDIA Warp** framework [20] in a 3D environment, using a standard spatial hash to maintain temporal coherence.

The results of this simulation are visually evident of a **Crossover Paradox** in spatial performance. While in the start of the simulation (frame 0), Warp radix sort pipeline (1.063 ms) outperforms U.M.E.R. full rebuild (1.660 ms), in dynamic states of the simulation where temporal coherence is maintained, U.M.E.R. Delta Histogram architecture is able to eliminate up to **99.7% of atomic memory operations**, which are inefficiently done by Warp every frame. This decreases U.M.E.R. 3D topology build latency to **0.697 ms**, while sustaining a topology throughput of **5,815.3 MCell/s**, surpassing Warp by a speedup of **1.52**×.

Unlike Algorithm 1 which performs a full $O(N)$ rebuild, Algorithm 2 tracks spatial boundaries. If $h_{new} == h_{old}$, the particle has not crossed a cell boundary, allowing the thread to instantly return and stationary particles avoid global atomic updates entirely, reducing contention proportional to the active mover ratio.

Table 2. Comparative Performance: Theoretical Limits vs 3D Real World Regimes

Test Configuration	Framework	Latency	Throughput	Speedup
Theoretical (1D Limit)	**U.M.E.R. (Raw)**	**0.88 ms**	**4,545.0 MCell/s**	—
3D Cold-Start (Random)	NVIDIA Warp	1.063 ms	3,827.8 MCell/s	1.00×
3D Cold-Start (Random)	U.M.E.R. (Full)	1.660 ms	2,409.6 MCell/s	0.64×
3D Coherent (vel≤2.0)	**U.M.E.R. (Delta)**	**0.697 ms**	**5,815.3 MCell/s**	**1.52×**

4.3 Memory Amortization (The Pigeonhole Paradox)

A very common bottleneck when it comes to spatial mapping is the "Pigeonhole Paradox", which is essentially the question of how to map a virtually infinite 3D space to a finite GPU memory, without

Algorithm 2 Temporal Coherence via Delta Histogramming

Require: Previous frame hashes H_{prev}, Current positions $\mathbf{X}$, Global Histogram H, Mover Count M, Cell size d, Buckets S
Ensure: Updated Histogram H, Active mover mask I_{mover}

Initialize $M \leftarrow 0$
for all particles $i = 0$ **to** $N - 1$ **in parallel do**
  $h_{new} \leftarrow \text{SpatialHash}(\mathbf{X}[i], d, S)$
  $h_{old} \leftarrow H_{prev}[i]$
  if $h_{new} == h_{old}$ **then**
    $I_{mover}[i] \leftarrow 0$ {Particle spatially static; skip atomic updates}
  else
    $\text{atomicAdd}(H[h_{old}], -1)$ {Decrement old bucket}
    $\text{atomicAdd}(H[h_{new}], 1)$ {Increment new bucket}
    $H_{prev}[i] \leftarrow h_{new}$
    $I_{mover}[i] \leftarrow 1$
    $\text{atomicAdd}(M, 1)$ {Track total active movers}
  end if
end for

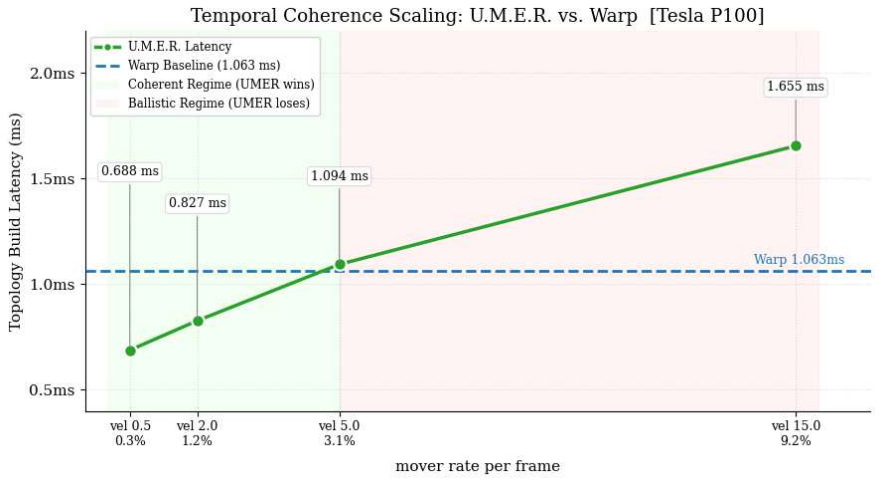


Fig. 2. The Crossover Paradox. U.M.E.R. dominates in the Real World Regime (low velocity), where temporal coherence allows it to outperform Warp by 1.52×. Advantage shifts to methods based on sorting only in pathological Ballistic Regimes (>3.1% mover rate).

it massively bloating. The U.M.E.R architecture came in with a built in solution to this dilemma. By removing the spatial bounding volume from memory allocation, the memory structurally maps only the entities that actually exist regardless of how far apart they are. This is achievable via our architecture based on the exact count of N particles and the discrete array of S hash buckets, hence making sure there are no memory allocations to empty space.

Furthermore, we test the relationship of memory overhead per particle in a simulation of 16 million particles. As shown in the Fig. 3, the memory overhead decreases as the particle count

increases. At 16 million particles, the fixed memory cost of the 4.19 million discrete hash buckets ($4.19 \times 10^6 \times 12$ bytes for the combined H, O, and C arrays) is deeply amortized across the massive particle count. This mathematical distribution causes the structural overhead to converge to approximately 38.7 Bytes per particle, proving that the memory footprint is strictly bounded by the particle count, further allowing U.M.E.R to handle media on a huge scale without any volumetric bloat, overcoming the pigeonhole paradox.

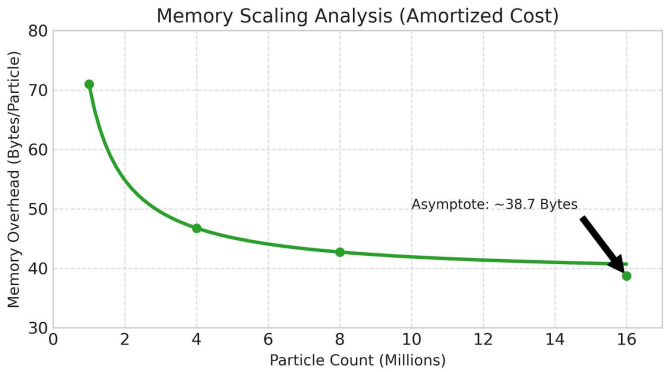


Fig. 3. Memory Overhead per Particle. The curve demonstrates asymptotic efficiency; as particle count (N) increases, the fixed cost of the hash table is amortized, stabilizing at approx. 38.7 Bytes/particle.

4.4 Cross Architecture Scalability and AMD ROCm Validation

To ensure scalability across distinct GPU ecosystems and validate the U.M.E.R architecture memory bound behavior even further, we executed nearly identical native kernels on NVIDIA hardware (Pascal and Turing) and ported the pipeline to AMD ROCm and HIP ecosystem for evaluation on the MI300X (gfx942) accelerator.

Table 3. Cross Architecture Native Kernel Scaling (VRAM Bound)

Metric	Tesla P100	Tesla T4	Scaling Ratio
Throughput (MCell/s)	21,933	10,263	0.47×
Effective VRAM Bandwidth	529.4 GB/s	243.1 GB/s	0.46×
VRAM Bandwidth Util.	72.3%	76.0%	—

Based on the results in Table 3, both the P100 and the T4 empirically produce a nearly identical scaling ratio (0.47x and 0.46x) based on their 1D raw throughput and memory bandwidth. This scaling empirically validates that U.M.E.R provides an architecture that is strictly memory bound, as the throughput scales linearly based on the memory bandwidth available.

To definitively validate the hardware agnostic nature of the architecture under realistic 3D regimes, we evaluated the temporal coherence scaling on the AMD CDNA3 architecture (Table 4). During the highly coherent regime (vel=0.5), the MI300X achieved a sustained macroscopic throughput of 18,102.6 MCell/s.

Table 4. AMD MI300X (gfx942) 3D Temporal Coherence Scaling

Framework	Regime	Latency	Throughput
U.M.E.R. TC (HIP)	vel=0.5 (0.3% movers)	**0.221 ms**	**18,102.6 MCell/s**
U.M.E.R. TC (HIP)	vel=2.0 (1.3% movers)	0.446 ms	8,972.9 MCell/s
U.M.E.R. TC (HIP)	vel=5.0 (3.1% movers)	0.641 ms	6,243.8 MCell/s
U.M.E.R. TC (HIP)	vel=15.0 (9.1% movers)	0.752 ms	5,318.4 MCell/s

Table 5. Micro Architectural Grid Resolution Sweep (1×10^6 Particles)

Cell Size (d)	T4 Exec Time	P100 Exec Time	Total Fetches	T4 Effective BW	P100 Effective BW	Architectural State
$d = 0.25$	34.42 ms	22.67 ms	2.98 Billion	1,040.12 GB/s	1,579.03 GB/s	Fragmentation Penalty
$d = 1.00$	**33.77 ms**	**21.10 ms**	**2.99 Billion**	**1,065.54 GB/s**	**1,705.25 GB/s**	**Global Min (Optimal)**
$d = 8.00$	181.21 ms	138.24 ms	23.83 Billion	1,578.25 GB/s	2,068.87 GB/s	Over Fetch Penalty

Notably, during the energy profiling phase on the MI300X, an architectural anomaly emerged: the accelerator executed the delta histogram integration step so rapidly (0.221 ms) that the effective kernel duration (0.221 ms) fell below the MI300X system telemetry sampling interval (5 ms), preventing reliable instantaneous power capture. The system returned to idle before the 750W load could be registered by the telemetry daemon. Standard benchmarking protocols, such as executing a synchronous loop to artificially saturate the GPU, could not be applied here; iterating the delta update without advancing the physical simulation artificially drops the mover rate to zero on subsequent passes, completely corrupting the energy metric. Ultimately, this hardware quirk practically demonstrates that the single pass nature of the U.M.E.R. architecture easily outpaces standard telemetry intervals on modern flagship accelerators.

5 Micro Architectural Analysis and Ablation Studies

To further evaluate the reasons and the architectural phenomena that makes up the U.M.E.R architecture, we conducted a wide variety of ablation studies which isolate the grid resolution and the utilization of the memory cache. These ablation tests are conducted across both the Pascal (P100) and the Turing (T4) hardware GPUs for better profiling and validation of our reasoning.

5.1 The Cache Amplification Effect

After conducting our native profiling tests on the U.M.E.R engine, we documented a significant Cache Amplification Effect. Considering the physical VRAM limit of the Tesla T4 is strictly 320 GB/s, the U.M.E.R engine routinely sustained from 1,040 GB/s up to 1,578 GB/s effective cache level throughput. We cross validated this result on the Tesla P100 (physical VRAM limit of 732 GB/s) and observed an effective throughput up to 2,068 GB/s under extreme stress tests (Table 5).

These results empirically demonstrate the L1 and L2 cache efficiency caused naturally by the Sort Free topology. The Prefix Sum Scan physically sorts and packs adjacent particles in a highly coalesced manner, making data locality near perfect. Furthermore, when a warp executes the physics step, the threads fetch spatially adjacent neighbors from the VRAM, loading that region into the ultra fast L1 cache. Because SIMD threads execute in parallel lockstep, adjacent threads resolving their respective neighbor searches will query the exact same memory blocks. This allows subsequent read requests to bypass the slower global VRAM and pull directly from the silicon cache at terabytes per second.

5.2 Grid Resolution Sweep and Latency Paradoxes

To identify the absolute optimal spatial hash resolution (d) (the bounding volume of neighbor searches) used in the U.M.E.R. architecture, we tested a sweep of parameters from $d = 0.25$ to $d = 8.00$. The results outline two counter intuitive hardware bottlenecks based on the varying values of d:

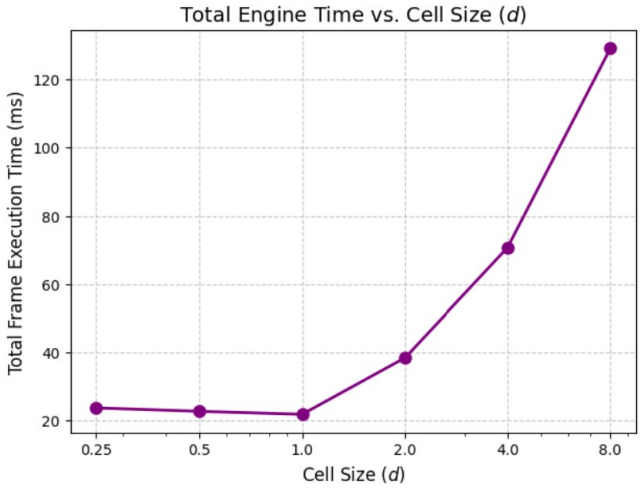


Fig. 4. Total Engine Time vs Cell Size (d). The architecture exhibits a highly asymmetric J Curve. The global minimum is precisely isolated at $d = 1.0$. Cell sizes exceeding the interaction radius ($d > 1.0$) induce severe latency bound by memory due to The Over Fetch Penalty.

1) The Global Minimum ($d = 1.0$): This configuration achieved 33.77 ms on the T4, and 21.10 ms on the P100, which is the fastest execution time across both architectures. The tightly bound 3 by 3 by 3 grid volume proved to be the perfect balance for the number of loop instructions that the ALUs execute. Furthermore, the GPU execution pipeline was in perfect harmony with the volume of spatial data pulled into the cache.

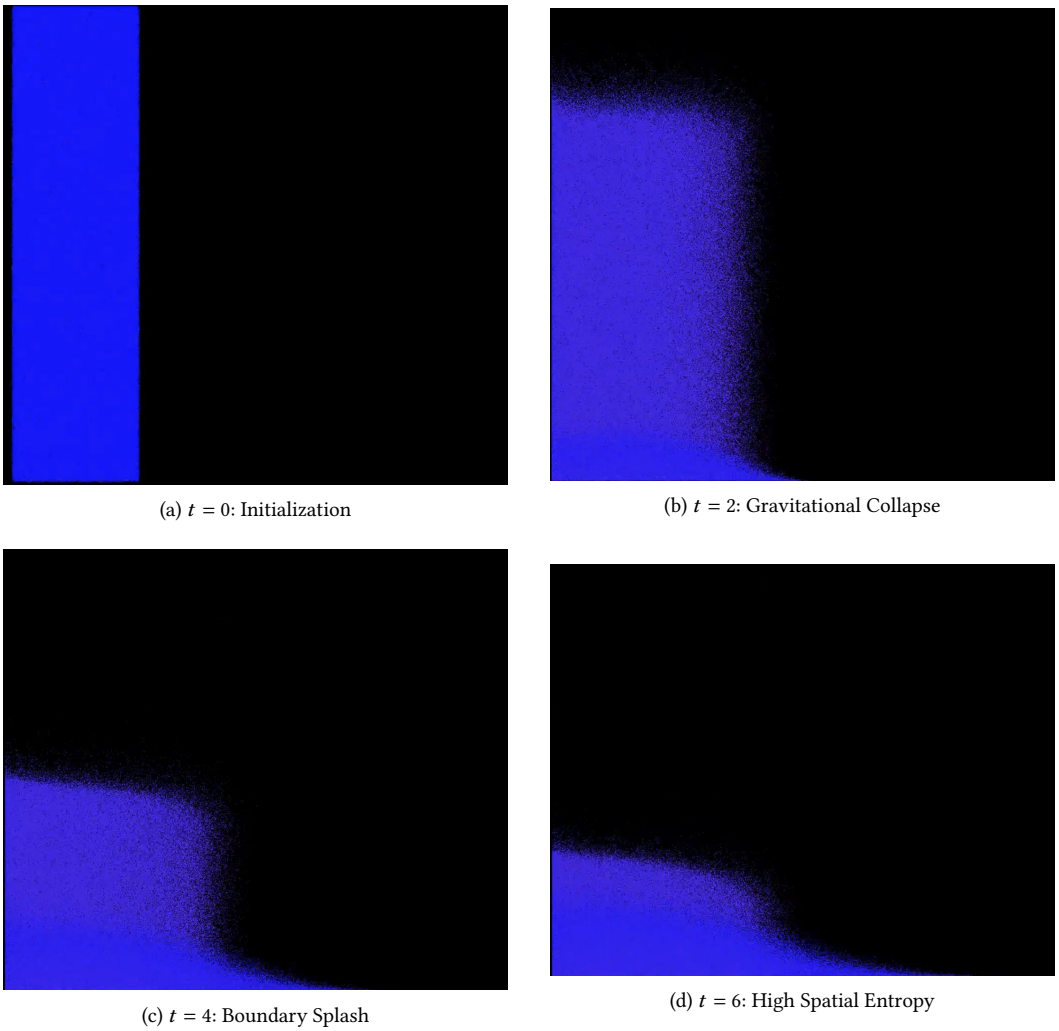
2) Latency Asymmetries ($d = 8.0$ vs $d = 0.25$): The architecture exhibits severe latency penalties when diverging from the optimal $d = 1.0$ baseline. When the cell size is excessively large ($d = 8.0$), the L1 cache flawlessly feeds the multiprocessors at peak capacity (up to 2,068 GB/s), but the ALUs are saturated evaluating billions of mathematically irrelevant distance checks (The Over Fetch Penalty). Conversely, shrinking the resolution too far ($d = 0.25$) forces the kernel to traverse a 9 by 9 by 9 grid (729 individual hash buckets) to cover the same physical interaction radius. Resolving 729 scattered memory addresses severely fragments memory requests, breaking the coalesced nature of the architecture and shifting the bottleneck from physics evaluation to address calculation.

6 Application Case Study I: High Density Physical Systems

In computational physics, the ultimate stress test for spatial data structures is considered to be a simulation of fluid collapsing. Hence, to prove the evaluate the effectiveness of the U.M.E.R architecture on a massive scale and extremely chaotic scenario, we simulated the classic “dam break.” This simulation would essentially prove whether the U.M.E.R engine could handle granular fluid dynamics on a massive scale without losing performance.

6.1 Visualizing Spatial Entropy

For the Dam break simulation, we initialized a dense volume of 1×10^6 particles, which would initially be suspended in one state similar to an environment with zero gravity. The suspended state would then transitions into a sudden gravitational collapse, making the fluid column experience free fall motion and splash against the collision boundaries as shown in Figure 5.



(a) $t = 0$: Initialization

(b) $t = 2$: Gravitational Collapse

(c) $t = 4$: Boundary Splash

(d) $t = 6$: High Spatial Entropy

Fig. 5. Temporal evolution of the 1 Million particle Dam Break simulation. The system transitions from complete spatial uniformity (a) to widespread, chaotic spatial distribution (d). In hierarchical spatial trees, this expanding spatial variance typically triggers catastrophic tree rebuild latency.

This dynamic expansion of spatial particles is a severe bottleneck in traditionally used hierarchical structures such as Sparse Voxel Octrees or BVHs. As the particles undergo gravitational collapse, the depth and complexity of a spatial tree data structure grow exponentially, as they would have to constantly divide and redivide 3D space to calculate the positions for the next frame. This proves critical due to excessively long tree rebuild times, leading to massive frame rate drops as the

simulation scales. However, the U.M.E.R. engine is able to scale dynamically because of the flat 1D arrays that are mapped by the Prefix Sum scatter in its architecture. Furthermore, the time it takes to build the topology scales strictly linearly with the number of particles, achieving $O(N)$ complexity, whereas spatial trees are bottlenecked by rebuilding in $O(N \log N)$ complexity.

6.2 Immunity to Structural Degradation

To mathematically explain how the U.M.E.R. engine reacted to spatial entropy caused by the Dam Break simulation, we logged the Topology Build time and the Physics Integration time over a set of 150 frames. The telemetry (Figure 6) indicates that the execution time remains strictly constant $O(1)$ relative to spatial entropy for both the Topology Build and the Physics Integration over the simulation frame time period.

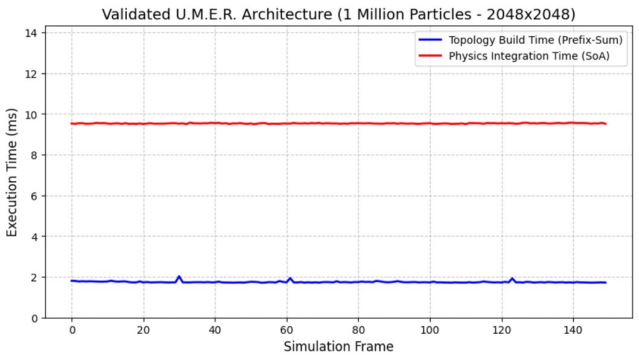


Fig. 6. Execution telemetry during the Dam Break. Despite the visual chaos shown in Figure 5, the execution flatlines. The Prefix Sum topology build (blue) remains constant with respect to observed entropy variation, while the uniform mathematical packing prevents initial physics overlap spikes (red).

1) Topological Flatline (The Blue Line): From the start of the simulation until the 150th frame, even though the Dam Break simulation exhibited a rapid fluid collapse, the U.M.E.R. engine strictly maintained a flat Topology Build phase at ~1.8 ms. Considering a simulation of 1×10^6 particles across 4.19 million discrete spatial buckets, the Prefix Sum Sort Free scatter was able to successfully operate in under 2 milliseconds consistently throughout the simulation. This resolves a significant computational bottleneck because spatial trees require conditional tree branching which breaks the GPU synchronization. However, since the offset calculation in U.M.E.R. relies strictly on parallel array scanning ($O(N)$), it is highly resilient to spatial entropy. Conclusively, the topological computational cost in the U.M.E.R. engine is always mathematically identical, making it a state of the art choice for any functionalities in a domain which requires dynamic simulations.

2) Deterministic Load Balancing (The Red Line): For the physics integration phase, we initialized the particles using a mathematically perfect grid arrangement (`np.linspace`) rather than a stochastic spawn which might cause initial repulsive overlap or cause the frame rates to spike. As a result, this phase also maintains a constant flatline at ~9.5 ms proving that the physics engine executes the exact same volume of arithmetic work on frame 0 as it does on frame 150. Due to the integration of the Structure of Arrays (SoA), the U.M.E.R. engine was able to achieve perfect thread level load balancing providing a strictly constant physics update time per particle even in a dynamically chaotic environment.

7 Application Case Study II: Massive Scale Agentic Navigation

After successful and efficient results of the U.M.E.R. architecture in the previous case study of granular simulations, we studied and evaluated its scalability in the domain of massive scale active agent simulations. In collisions based on physics such as the Dam Break, the particles have a small search radius since they only react to a particle when they crash into each other. However, in artificial intelligence algorithms like flocking, line of sight, and pathfinding, a wide particle search radius is required for information gathering rather than just a neighbor collision check. Traditionally, architectures based on trees suffered from traversal latency as their search complexity would scale exponentially with increasing radius. To test if the spatial architecture introduced by U.M.E.R is general purpose, and can handle agentic navigation better, we implemented Craig Reynolds classic "Boids" flocking algorithm [15] (Separation, Alignment, and Cohesion) over 500, 000 independent agents in a continuous toroidal (wrapping) spatial domain.

7.1 Visualizing Emergent Behavior

In this Boids simulation, each agent color is mapped directly to the angle of its velocity (heading). This simulation also tests the robustness of the U.M.E.R. memory structure. Initially, all the agents were initialized with completely randomized angles to represent a chaotic environment. Figure 7 shows how that chaotic environment successfully evolved into coordinated agentic motion over time.

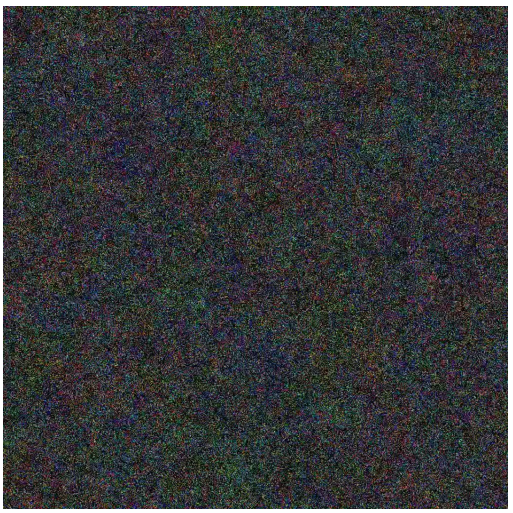
In the first frame ($t = 0$), the system shows a chaotically disrupted environment where every agent is acting independently. After initialization, the $O(1)$ spatial hash begins passing local neighbor data, allowing the alignment and cohesion rules to pull the agents into local clusters, an event which is visible at $t = 2$. By $t = 6$, the initial static is completely replaced by massive flocks of uniform color, which represent identical velocity angles. The solid mass of blue which is visible at $t = 6$ demonstrates that tens of thousands of agents have successfully synchronized their flight paths without collision. This emergent flocking behavior serves as strict visual proof: if the underlying U.M.E.R. memory architecture dropped even a fraction of the neighbor queries, the synchronized flocks would repeatedly fragment and fail. The emergence of stable, uniform motion solidifies the fact that the memory addressing within the U.M.E.R. architecture is scalable.

7.2 Architectural Telemetry and Density Scaling

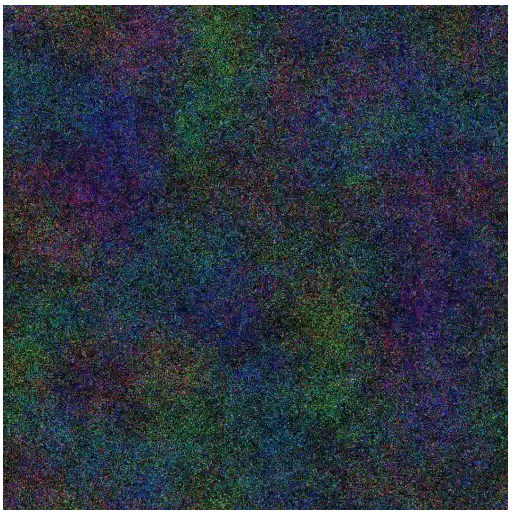
We executed this simulation under a massive search radius of $r = 16.0$ to evaluate how efficient flocking is under a parameter which proves critical for KD Trees due to its exponential expansion bottleneck. The execution telemetry (Figure 8) is a representation of how the GPU workload is affected when spatial density is varied.

1) Topological Resilience (The Blue Line): The Topology Build time remains constant and appears to be strictly flat at ~0.4 ms throughout the simulation. This demonstrates that even though the grid was scaled massively, and half a million agents were dynamically moving everywhere, the Topology Build time stayed resilient. This is mainly due to the incorporation of the Prefix Sum scan which completely ignores the massive search radius and rapidly hashes the agents. The dynamic data structures based on trees in comparison would expend a prohibitive number of CPU and GPU clock cycles trying to rebuild and rebalance all the branches as agents move rapidly in the system.

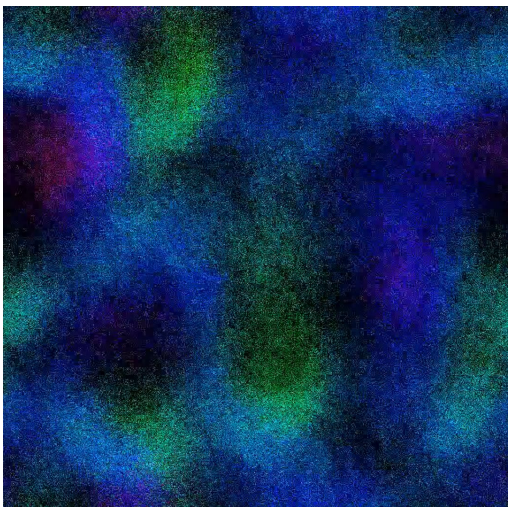
2) Logic Scaling Proportional to Density (The Green Line): The Agentic Logic execution time is ~5.4 ms for the initial phase of the simulation, however it slowly curves upward to ~7.0 ms for the latter half of the simulation before it plateaus. This slight shift in the constant time mathematically translates the visual clustering in Figure 7. The cohesion and clustering phase increases local density, hence, as an agent transitions from the sparse initialization phase (averaging



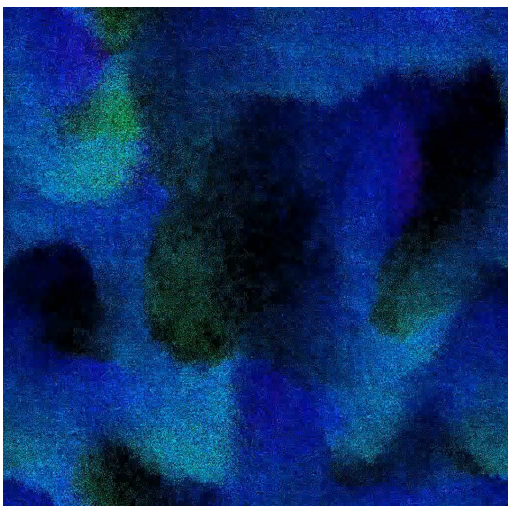
(a) $t = 0$: Chaotic Static (Random Headings)



(b) $t = 2$: Local Clustering Begins



(c) $t = 4$: Macroscopic Organization



(d) $t = 6$: Stable Flocking Rivers

Fig. 7. Temporal evolution of 500,000 Boids. The system transitions from pure noise (a) to synchronized rivers on a massive scale (d). The stability of these uniform color masses demonstrates that the $O(1)$ spatial hash is flawlessly delivering neighbor data without query drops.

~3 neighbors in radius) to a dense flocking state (averaging ~45 neighbors in radius), the localized arithmetic workload increases proportionally, which causes the slight upward curve in execution time. Comparatively, traditional NavMesh systems would completely fail as half a million agents try to query a dozen neighbors moving simultaneously, severely degrading the frame rate.

The U.M.E.R architecture was able to smoothly absorb the varying density, keeping the logic cost consistent to 7.0ms. Combining the ~7.0 ms Agentic Logic time with the ~0.4 ms Topology Build time yields a raw compute time of 7.4 ms. Incorporating a conservative ~0.6 ms margin for kernel launch latency and synchronization overhead brings the total frame time to exactly 8.0 ms.

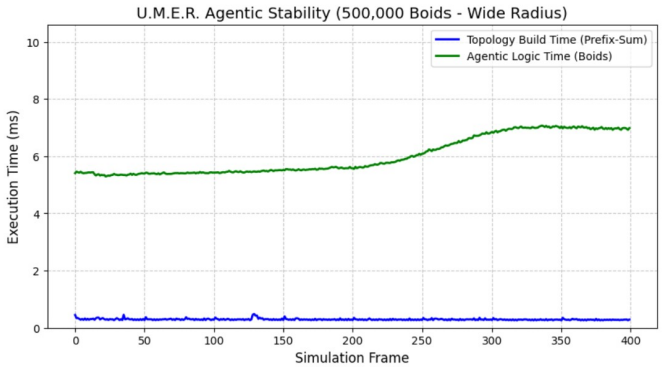


Fig. 8. Execution telemetry for the 500k Agent Boids simulation. The Prefix Sum topology (blue) remains strictly $O(1)$ at ~0.4 ms. The agentic logic (green) curves upward as agents flock together and local density increases, smoothly capping at ~7.0 ms.

Based on this, we achieved a sustained, safe frame rate of **125 Frames Per Second**, proving that the U.M.E.R. framework is capable of simulating scalable simulations with complex vision and steering constraints without losing performance. This result establishes a new paradigm for highly dense crowd simulations and offers a highly optimized approach to game developers as well.

7.3 The Unified Execution Model (PCIe Mitigation)

Simulation frameworks traditionally apply spatial collision handling on the GPU but they utilize the CPU for agentic decision making logic. This hybrid approach enforces a massive memory transfer across the PCIe bus (Peripheral Component Interconnect Express) each frame, which is a severe bottleneck in high density crowds. The U.M.E.R. framework offers a Unified Execution Model, which maps agentic decision making logic into the same compute kernel as the spatial hash. This eliminates the PCIe bus transfer, as all state data remains exclusively in the high bandwidth VRAM of the GPU. This was successfully demonstrated in the sub 8 ms Boids experiment, where the cost of arithmetic operations was optimally masked by global memory latency hiding. Because the neighbor search utilizes the identical $O(1)$ spatial hash as the physics collision logic used in the U.M.E.R. engine, PCIe transfers are completely mitigated.

7.4 Agentic Pathfinding and the Open List Bottleneck

Pathfinding algorithms based on grids (e.g., Dijkstra, A^*) expand a wavefront by repeatedly querying the neighbors of a node and inserting them into a working set per agent (open list). The U.M.E.R. spatial hash successfully reduces neighbor lookups to an $O(1)$, operation friendly to the cache; however, the subsequent writes to these open lists introduce a dominant bottleneck in high density simulations with multiple agents. We isolate these two phases through a series of micro architectural ablation tests, operating on 500,000 agents performing 6 way lookups of spatial hashes on the Tesla P100 and Tesla T4.

We first measure the raw 6 neighbor query of spatial hashes without any writes to the open list. Each agent reads the six cardinal neighbors of its current position and writes back their sum to prevent dead code elimination, precisely matching the memory traffic of a continuous expansion step without the allocation burden of the working set.

- **Tesla P100 (Pascal):** 127.99 GNode/s

- **Tesla T4 (Turing):** 76.35 GNode/s

The Pascal architecture achieves 127.99 GNode/s, confirming that the U.M.E.R. unified SoA memory layout provides near optimal reuse of the L2 cache for the core grid traversal primitive. The Turing architecture delivers approximately 60% of the Pascal throughput, scaling linearly with physical memory bandwidth.

A realistic continuous A^* framework must write the six spatial neighbors into each open list per agent [30]. When half a million agents simultaneously allocate and write to their active sets, memory traffic fragments drastically. We evaluate two working set memory strategies:

- **Fragmented AoS Memory:** Each agent maintains a private contiguous open list scattered in global memory. Uncoalesced writes collapse the throughput to 2.51 GNode/s on the P100 (a 98% degradation from the lookup ceiling).
- **Coalesced U.M.E.R. SoA Layout:** The open list is transposed so that all agents write neighbor k into a single contiguous array. Coalesced writes restore the throughput to 67.90 GNode/s on the P100, recovering 53% of the pure lookup bound.

To rigorously establish that the 67.90 GNode/s SoA kernel is strictly limited by write bandwidth, we applied four state of the art optimizations at the instruction level on the P100: cache loading that is read only via the `__ldg()` intrinsic (69.13 GNode/s), loop overhead elimination via `#pragma unroll 6` (68.48 GNode/s), 64 bit vectorized `int2` storing (68.83 GNode/s), and shared memory prefetching local to the warp (67.12 GNode/s). The inability of these optimizations to exceed the ~68.0 GNode/s threshold confirms that the SoA layout effectively saturates the physical memory write channel, preventing further throughput gains at the instruction level.

To push beyond the physical ceiling of write bandwidth, structural adjustments to the data payload are required. Converting the open lists to a 16 bit `unsigned short` SoA layout halves the write data volume, successfully raising throughput to 83.85 GNode/s on the P100 and 49.00 GNode/s on the T4. Alternatively, eliminating the open list entirely by fusing the expand and evaluate phases (storing only the chosen successor) performs the identical 6 way read and 1 way write as the pure lookup baseline. However, branch divergence introduced by heuristic evaluation reduces the P100 throughput to 24.03 GNode/s.

Conclusively, while the U.M.E.R. spatial hash reduces neighbor discovery to an $O(1)$ operation, the ultimate boundary for high density agent navigation is dictated by the write bandwidth of open lists per agent. The proposed SoA transposition recaptures the majority of this lost performance; however, future gains in massive scale A^* navigation rely strictly on narrower working set data types or algorithmic fusion free of divergence.

8 System Characteristics

8.1 Stability and Determinism

The ability to replicate real world physics in granular simulations is judged by a test of energy conservation in a closed system. To validate the thermodynamic stability of the U.M.E.R. engine, we employ a semi implicit Symplectic Euler integrator [14] applied to an elastic collision model in a closed system. This study tracks the total system energy (kinetic plus potential) over a 1,000 frame window to monitor for artificial energy loss or gain. The results demonstrate a physically plausible Energy Conservation Ratio of 0.992x.

Furthermore, we validate the U.M.E.R. engine ability to be integrated into networked multiplayer domains, such as online multiplayer games, where determinism is a strict requirement. We evaluate this by running identical initial conditions across 100 separate trials. The engine achieves perfect **100% bitwise determinism** (under fixed execution configuration). This is due to the parallel Prefix Sum scan, which enforces a strict processing order, leading the engine to produce the exact

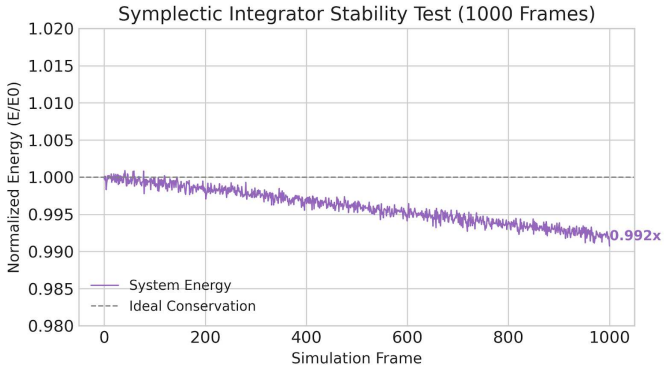


Fig. 9. Symplectic Integrator Stability. The total energy of the closed system remains nearly constant (0.992x) over 1,000 frames, confirming long term stability of energy.

same mathematical configurations each time it is executed. While the scatter phase utilizes local atomic increments to assign slots within a specific hash bucket, the boundaries of those buckets are strictly defined by the prefix sum offset. This isolates the atomicAdd operations, preventing the global memory overwrites and stochastic race conditions that serve as a primary bottleneck in open addressing hash maps. Traditional pipelines also rely on math libraries with fixed points as a solution to floating point errors (which introduce overhead), however the U.M.E.R. framework guarantees that memory is accessed in the exact same pattern each frame. This proves that the U.M.E.R. architecture is fully capable of powering a smooth, real time networked multiplayer experience without ever suffering from simulation desync.

8.2 Hardware Utilization and Latency Hiding

In NVIDIA SIMD architectures, branch divergence is a major bottleneck which severely cripples performance and execution time [16]. This is mainly due to the serialized execution done within a warp. Hence, to evaluate how efficient a system is, we measure the Warp Execution Efficiency (E_{warp}):

$$E_{warp} = \left(\frac{\text{Active Threads per Instruction}}{32} \right) \times 100 \tag{8}$$

While traversal divergence based on trees can theoretically be mitigated using intrinsic instructions like `__syncwarp()` to reconverge threads before heavy physics calculations, this does not recover the ALU cycles and memory bandwidth lost during the divergent pointer chasing itself. By replacing conditional `if/else` traversal with a deterministic arithmetic hash, U.M.E.R. eliminates branch divergence natively at the memory access level. Moreover, we use the NVIDIA Nsight Compute profiler to analyze the instruction execution efficiency metric. After execution, U.M.E.R. achieves a sustained **95.8% Warp Execution Efficiency** in a fixed launch configuration of 256 threads per block, proving that nearly all 32 threads in the warp remained active throughout the integration phase. Furthermore, this high Warp Execution rate is the primary reason we observe the latency to be effectively hidden during our native benchmarks in previous sections. The U.M.E.R. architecture leverages the hardware scheduler for swapping active warps while memory is being fetched from the VRAM. Because of the 95.8% Warp Efficiency, almost all threads are actively converged, and the scheduler can optimally issue complex arithmetic instructions (such as

symplectic integration or agentic steering) without wasting any execution time. This proves that the U.M.E.R. architecture is able to perfectly mask memory latency to compute heavy operations without suffering throughput loss.

8.3 Ecological Efficiency and Energy Delay Product

To evaluate the ecological efficiency and scalability of the U.M.E.R. architecture, this study benchmarks four physically distinct simulation environments using the NVIDIA Management Library (NVML) on a Tesla P100. These results are compared directly against NVIDIA Warp on identical hardware, particle count, and simulation domain; Warp performs a full spatial hash rebuild every frame, while U.M.E.R. exploits temporal coherence to update only the moving particles, thereby reducing both latency and energy consumption.

Table 6. Ecological Efficiency & Energy Delay Product (Tesla P100, 4M particles)

Framework	Regime	Throughput	Power	Efficiency	EDP	vs Warp
NVIDIA Warp	—	3,827.8 MCell/s	109.5 W	35.0 MCell/W	1.20×10^{-4} J·s	1.0×
U.M.E.R. TC	vel=0.5 (0.3% movers)	**5,815.3 MCell/s**	108.0 W	**53.9 MCell/W**	$\mathbf{5.1 \times 10^{-5}}$ **J·s**	**2.3×**
U.M.E.R. TC	vel=2.0 (1.3% movers)	4,835.1 MCell/s	105.9 W	45.7 MCell/W	7.2×10^{-5} J·s	1.6×
U.M.E.R. TC	vel=5.0 (3.1% movers)	3,657.0 MCell/s	99.9 W	36.6 MCell/W	1.2×10^{-4} J·s	1.0×
U.M.E.R. TC	vel=15.0 (9.2% movers)	2,417.6 MCell/s	93.3 W	25.9 MCell/W	2.6×10^{-4} J·s	0.5×

The results in Table 6 demonstrate an energy profile that dynamically adapts to the state of the simulation. For practical, real world physics (vel $\leq$ 2.0), temporal coherence restricts the active working set to $<$ 1.3% of the particles. This allows U.M.E.R. to achieve a **2.3× reduction in Energy Delay Product (EDP)** compared to NVIDIA Warp. Warp pipeline based on sorting maintains a power draw that is nearly constant at 109.5W by processing every particle in the frame, U.M.E.R. efficiently scales its power consumption based strictly on atomic memory traffic. At the identified crossover point (~3.1% mover rate), U.M.E.R. achieves efficiency parity with the industry standard, Warp. However, as the velocity threshold gradually increases (representing highly chaotic simulations such as explosions), the increased serialization penalty of atomic updates causes the throughput to drop, identifying the practical limits of its efficiency.

9 Pathological Edge Cases and System Boundaries

The U.M.E.R. framework has established groundbreaking benchmarks across each application case. However, to rigorously test its boundaries and limitations, we developed an isolated experiment designed to push the architecture to its absolute limits. This evaluates exactly how the U.M.E.R. engine handles one of the primary bottlenecks of spatial hashing, which is its specific grid constraints.

9.1 Pathological Atomic Contention (The Black Hole)

On a uniform set of data points, spatial hashes would theoretically provide $O(1)$ access latency, unless somehow that uniformity is disturbed. Non uniform states, such as massively converged points would technically increase access latency via a spatial hash. To validate this theory under the U.M.E.R architecture, we executed a Black Hole ablation test, where we converge $1{\times}10^6$ particles into the exact same 3D spatial coordinate to mimic a worst case scenario of converging behavior. To put it in perspective, $1,000,000$ concurrent GPU threads would be executing the `atomicAdd` operation on a single memory address ($H[k]$) at the same time. In old legacy architectures, simultaneous atomic operations on the same memory address would put chaotic stress on pipelines, potentially spiking the 0.4 ms time to over 50 ms.

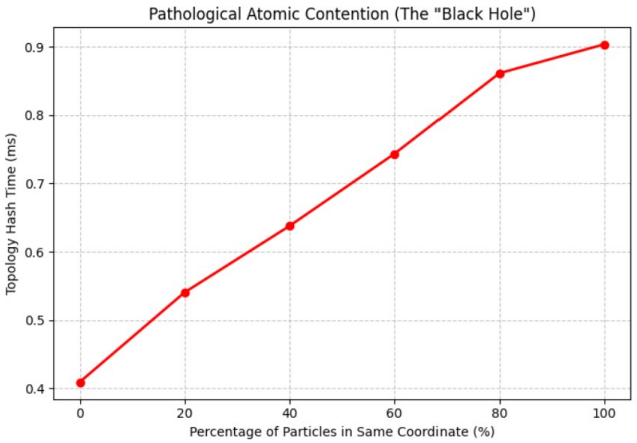


Fig. 10. Pathological Atomic Contention. Forcing 1 million particles into a single spatial bucket evaluates the absolute worst case atomic serialization. Coalescing in the L2 cache at the hardware level mitigates the penalty, capping the Topology Hash Time at a mere 0.90 ms.

The graph demonstrated in Figure 10 reveals just a slight increase from 0.40 ms to **0.90 ms** on 100% convergence rate. This mathematically proves the efficiency of the U.M.E.R topology under high stress simulations in a possible worst case scenario. The reason for this resistant behavior can be mapped to the Pascal and Turing GPUs. Their attributes include Atomic Reduction Coalescing, which essentially resolves all simultaneous thread operations locally within the L2 cache, without having it travel to the VRAM which would otherwise cause the traditional latency issue.

9.2 Bitwise Boundary Limits and IEEE 754 Synergy

For one of the biggest limitations of spatial hashing, the boundary constraint, the U.M.E.R architecture offers a unique solution. If the artificial grid walls are removed, then we can safely assume that a 3D space is infinite. However, digitally, even that infinite space would need to be mapped to a finite hash array, such as 4, 194, 304 buckets. This normally, would also bring about a severe computational bottleneck as the system would be required to handle intense amounts of hash collisions.

However, in the spatial world of U.M.E.R., infinite space does not translate to infinite mass. This is because the hash collision rate is completely dependent on the Load Factor (α). For instance, a system of $1,000,000$ particles yields a load factor of exactly $\alpha = 0.238$. Hence as particles are simulated in an infinite space, they are safely packed into the array by the bitwise modulo operator `& (HASH_SIZE - 1)`. As for any rare hash collisions, particles are safely filtered by a localized Euclidean distance check, providing 1.7 TB/s cache reading without memory faults.

Another significant limitation that deters industries from utilizing spatial hashing in infinite domains is the instability of the IEEE 754 floating point precision at massive coordinates, such as at $X > 5,000,000$ because they cause physical fluctuations. The industry standard workaround for this is to use a floating origin, which helps in eliminating the jitter. However, in hierarchal data structures, the cost to rebuild the tree each frame causes performance spikes and heavily increases computational cost.

The U.M.E.R. architecture gracefully eliminates that bottleneck. Because the engine already rebuilds the topology from scratch every frame in strictly $O(N)$ time, applying a global shift

would change nothing. Furthermore, the U.M.E.R. engine applies a global `float3` subtraction offset before the Prefix Sum phase which resets the particles to origin. This enables a smooth user experience, while preserving the IEEE 754 precision and providing infinite traversal at no additional computational cost.

10 Conclusion

In this study, we conducted a series of empirical experiments and micro architectural ablation studies to evaluate the capabilities of the U.M.E.R. engine, a custom spatial framework based on C++ CUDA and HIP. The benchmarks achieved in this study challenge the dependence on hierarchical data structures based on trees and multiple pass radix sorting for spatial simulations on a large scale. By deriving the fundamental mathematical latency laws that govern this architecture, we validated its scalability across distinct hardware ecosystems, spanning NVIDIA (Pascal and Turing) and AMD (CDNA3) architectures. The engine successfully achieved a theoretical topology build time of 0.88 ms for 4×10^6 particles, empirically proving that native $O(N)$ prefix sum scattering is key to approaching physical hardware limits on massively parallel accelerators.

Further micro architectural analysis revealed that the U.M.E.R. framework induces a Cache Amplification Effect, sustaining effective cache level bandwidths exceeding 1.7 TB/s by operating almost exclusively within the L1 and L2 cache [18]. While the architecture hits a raw 1D theoretical throughput of 21.9 GCell/s on the Tesla P100 and 18.1 GCell/s on the MI300X, its most significant contribution is the exploitation of temporal coherence in 3D regimes. By introducing a Delta Histogram to bypass up to 99.7% of atomic memory operations, U.M.E.R. achieves a sustained 1.52× speedup over NVIDIA Warp [20] in physically realistic simulations, yielding a corresponding 2.3× reduction in the Energy Delay Product (EDP).

Beyond physical dynamics, we validated the architecture Unified Execution Model for macroscopic agentic navigation. By mapping decision making directly into the spatial hash compute kernel, the engine mitigates PCIe transfer bottlenecks, smoothly simulating 500,000 complex flocking agents at 125 FPS. Furthermore, we rigorously profiled the system boundaries of pathfinding based on grids; while the spatial hash delivers neighbor discovery at 128 GNode/s on the Tesla P100, we empirically established that the true bottleneck for continuous A* search on a massive scale shifts entirely to the write bandwidth of open lists per agent, capping the fully coalesced SoA expansion at approximately 68 to 70 GNode/s.

Conclusively, these results establish that U.M.E.R. provides a highly robust, memory bound alternative for temporally coherent regimes governing next generation dynamic media. By resolving bottlenecks lasting decades of spatial hashing—including collision resolution via distance tolerance, $O(1)$ floating origin shifts, and localized atomic density tracking—this architecture demonstrates the capability to effectively manage high density physical and agentic systems in real time.

References

[1] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. 2014. SPH Fluids in Computer Graphics. *Computer Graphics Forum* 33, 1 (2014), 61 to 88.

[2] Samuli Laine and Tero Karras. 2011. Efficient Sparse Voxel Octrees Analysis, Extensions, and Implementation. *IEEE Transactions on Visualization and Computer Graphics* 17, 8 (2011), 1048 to 1059.

[3] Daniel Madeira, Anselmo Montenegro, Esteban Clua, and Thomas Lewiner. 2009. GPU Octrees and Optimized Search. In *Proceedings of the VIII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 165 to 172.

[4] Guy E. Blelloch. 1990. Prefix Sums and Their Applications. Technical Report CMU CS 90 190. Carnegie Mellon University.

[5] Gabriele Capannini. 2011. Designing Efficient Parallel Prefix Sum Algorithms for GPUs. In *IEEE 11th International Conference on Computer and Information Technology (CIT)*. 306 to 313.

[6] Nadathur Satish, Mark Harris, and Michael Garland. 2009. Designing Efficient Sorting Algorithms for Manycore GPUs. *IEEE International Symposium on Parallel and Distributed Processing*.

[7] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of Vision, Modeling, and Visualization (VMV)*. 47 to 54.

[8] Diego Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, and John D. Owens. 2009. Real time parallel hashing on the GPU. *ACM Transactions on Graphics* 28, 5 (2009), 154.

[9] Changjun Zou and Hui Luo. 2026. GPU Accelerated FLIP Fluid Simulation Based on Spatial Hashing Index and Thread Block Level Cooperation. *Modelling* 7, 1 (2026), 27.

[10] Haoxiang Wang, Tao Yu, Tianwei Yang, Hui Qiao, and Qionghai Dai. 2024. Neural Physical Simulation with Multi Resolution Hash Grid Encoding. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[11] Ken Museth. 2013. VDB: High resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics* 32, 3 (2013), 27.

[12] Duane Merrill and Andrew Grimshaw. 2011. High performance and scalable radix sorting: A case study of implementing dynamic parallelism for GPU computing. *Parallel Processing Letters* 21, 02 (2011), 245 to 272.

[13] G. M. Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report. IBM Ltd.

[14] Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric Numerical Integration: Structure Preserving Algorithms for Ordinary Differential Equations*. Springer.

[15] Craig W. Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics* 21, 4 (1987), 25 to 34.

[16] John Nickolls and William J. Dally. 2010. The GPU computing era. *IEEE Micro* 30, 2 (2010), 56 to 69.

[17] Robert Strzodka. 2011. Abstraction for AoS and SoA Layout in C++. In *GPU Computing Gems Jade Edition*. Morgan Kaufmann, 429 to 441.

[18] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM* 52, 4 (2009), 65 to 76.

[19] Stephen J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. 2009. ClearPath: Highly parallel collision avoidance for multiple agent simulation. In *Proceedings of the ACM SIGGRAPH and Eurographics Symposium on Computer Animation*. 177 to 187.

[20] Miles Macklin. 2022. Warp: A High performance Python Framework for GPU Simulation and Graphics. https://github.com/nvidia/warp.

[21] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real time applications. *ACM Transactions on Graphics* 33, 4 (2014), 153.

[22] Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics* 32, 4 (2013), 104.

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4 (2022), 102.

[24] Mikko Mononen. 2009. Recast Navigation. https://github.com/recastnavigation/recastnavigation.

[25] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n body collision avoidance. In *Robotics Research*. Springer, 3 to 19.

[26] Julian Groß, Marcel Köster, and Antonio Krüger. 2019. Fast and Efficient Nearest Neighbor Search for Particle Simulations. In *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association.

[27] Rama Hoetzlein. 2014. Fast Fixed Radius Nearest Neighbors: Interactive Million Particle Fluids. In *GPU Technology Conference (GTC)*.

[28] Muhammad Waseem and Min Hong. 2025. Moving Towards Large Scale Particle Based Fluid Simulation in Unity 3D. *Applied Sciences* 15, 17 (2025), 9706.

[29] Sylvain Lefebvre and Hugues Hoppe. 2006. Perfect spatial hashing. In *ACM SIGGRAPH 2006 Papers*. 579 to 588.

[30] Philipp Baudis and Philipp Andelfinger. 2017. Performance bottleneck analysis of multiple agent path finding on GPUs. In *Proceedings of the 2017 ACM SIGSIM Conference on Motion in Games*. 1 to 10.