Master Degree in Artificial Intelligence

Boids Simulation: Parallel Programming Optimization

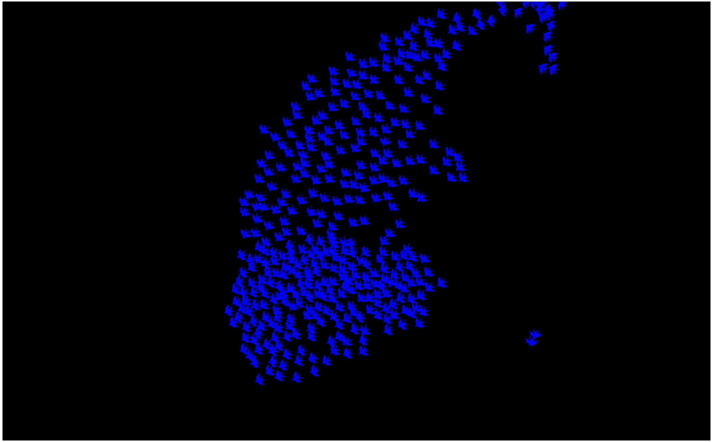
Parallel Programming for Machine Learning 2025

Gianni Moretti

Introduction

What are Boids?

- Introduced by Craig Reynolds in 1986
- Simulates flocking behavior of birds, fish, or other organisms



The Three Fundamental Rules

- **Alignment**: Each boid tries to match its velocity (direction and speed) to the average velocity of its local neighbors. This causes the group to move in a similar direction, promoting coordinated movement.
- **Cohesion**: Each boid steers towards the average position (center of mass) of its nearby flockmates. This keeps the group together and prevents individuals from straying too far from the flock.
- **Separation**: Each boid actively avoids crowding by steering away from neighbors that are too close. This prevents collisions and maintains a comfortable distance between individuals.

Each boid follows these rules independently, creating realistic flocking patterns without central control.

Project Goals

- Implement efficient Boids simulation in C++
- Compare sequential vs parallel implementations
- Analyze different data layouts: Array of Structures (AoS) vs Structure of Arrays (SoA)
- Evaluate performance scalability with OpenMP
- Utilize SIMD optimizations where possible

Implementation Overview

Technologies Used:

- **Language**: C++ for performance
- **Parallelization**: OpenMP for multi-core optimization
- **Visualization**: SFML (Simple and Fast Multimedia Library)
- **Build System**: CMake for cross-platform compilation

Implementation Details

Data Layout Comparison: AoS vs SoA

Array of Structures (AoS)

Traditional object-oriented approach

```cpp
struct BoidData {
    sf::Vector2f position;
    sf::Vector2f velocity;
    float biasval;
    int scoutGroup;
};
```

- Easy to understand and maintain
- Less cache-friendly for bulk operations
- Memory accesses can be scattered

Data Layout Comparison: AoS vs SoA

Structure of Arrays (SoA)

Performance-optimized layout

```
struct BoidDataList {
    float* xPos;
    float* yPos;
    float* xVelocity;
    float* yVelocity;
    float* biasvals;
    int* scoutGroup;
    int numBoid;
};
```

- Better cache utilization
- Enables SIMD vectorization
- Contiguous memory access patterns

Sequential Implementation

Main Algorithm Structure:

```
for (int i = 0; i < num_boids; i++) {
    // Compute alignment, cohesion, separation
    for (int j = 0; j < num_boids; j++) {
        if (i != j && distance < perception_radius) {
            // Apply boid interaction rules
        }
    }
    // Update position and velocity
}
```

Characteristics:

- $O(N^2)$ computational complexity
- Perfect candidate for parallelization

Parallel Implementation with OpenMP

Key Parallelization Strategy:

```
#pragma omp parallel for schedule(static)
for (int i = 0; i < N; i++) {
    // Each thread processes different boids
    // Thread-safe access to read-only data
    // Write to separate temporary arrays
}

#pragma omp simd
for (int i = 0; i < N; i++) {
    // Vectorized copy back to main arrays
    boidDataList.xPos[i] = new_xPos[i];
    boidDataList.yPos[i] = new_yPos[i];
    // ...
}
```

Parallel Implementation with OpenMP

Thread Safety Approach:

- Read-only access to current state
- Write results to temporary arrays
- Vectorized copy-back operation
- No race conditions or data dependencies

Compiler Vectorization Enablers:

- Contiguous memory access (SoA layout)
- `#pragma omp simd` directives
- Multiple data elements processed per instruction

Experimental Setup

Benchmark Methodology

Testing Environment:

- Multi-core CPU system (12 cores)
- Identical simulation parameters across all tests
- Multiple runs per configuration for statistical accuracy (30 runs)
- Isolated measurement of core algorithm (no rendering overhead)

Test Parameters:

- **Boid populations**: 1,000 to 32,000 agents
- **Thread configurations**: 1 to 12 threads
- **Implementations**: Sequential AoS vs Sequential SoA vs Parallel SoA

Performance Metrics

Primary Measurements:

- Execution time per simulation step
- Speedup vs sequential baseline
- Scaling efficiency with thread count

Analysis Focus:

- Impact of data layout optimization
- Parallel scalability characteristics
- Optimal thread configuration
- Performance vs problem size relationship

Computational Complexity Analysis

Algorithm Characteristics:

- **Time Complexity**: $O(N^2)$ per simulation step
- **Space Complexity**: $O(N)$ for boid storage
- **Parallel Potential**: Embarrassingly parallel outer loop

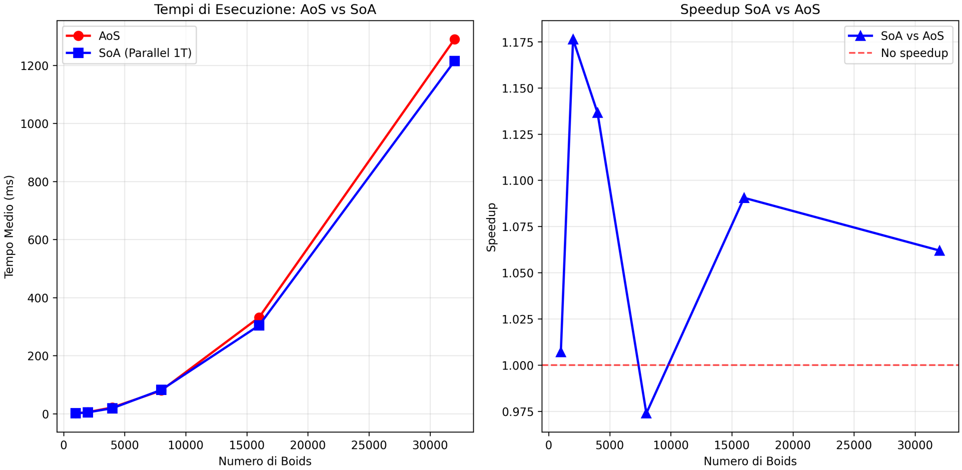
Performance Results

Data Layout Impact: AoS vs SoA

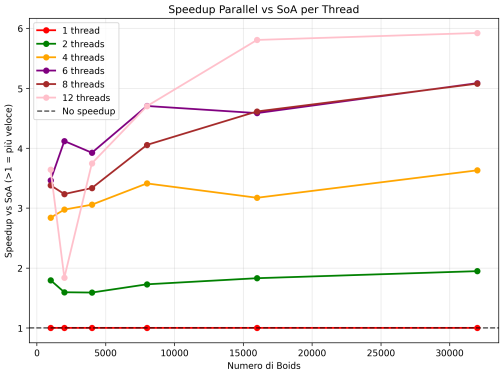
Key Findings:

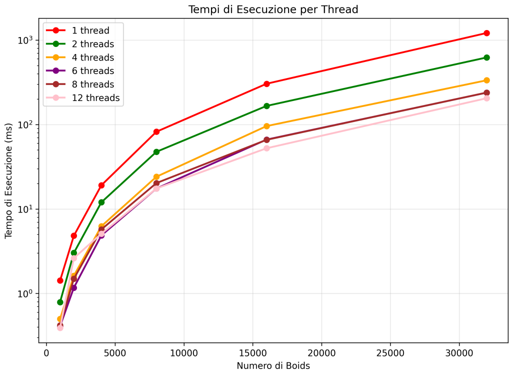
- SoA provides modest but consistent performance improvement
- 6.2% improvement for 32,000 boids (1,290.09ms vs 1,214.82ms)
- Benefits become more pronounced with larger datasets
- Foundation for effective parallelization

Data Layout Impact: AoS vs SoA



Parallel Scaling Performance





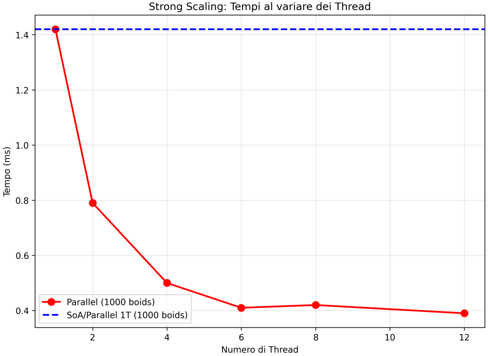
Parallel Scaling Performance

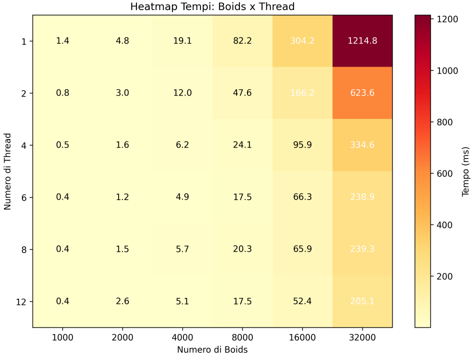
Speedup Results:

- Near-linear scaling up to 6 threads
- Diminishing returns beyond 8-12 threads due to overhead

Dataset Size	Sequential (ms)	Best Parallel (ms)	Speedup
1,000 boids	1.42	0.39	3.6x
4,000 boids	22.69	4.85	4.7x
8,000 boids	82.23	17.48	4.7x
16,000 boids	367.89	74.23	5.0x
32,000 boids	1,214.82	205.12	5.9x

Parallel Scaling Performance





Conclusions

Project Achievements:

- Successfully implemented and optimized Boids simulation
- Demonstrated significant performance improvements through parallelization
- Validated importance of data structure optimization
- Achieved up to 5.9x speedup on multi-core systems

Thanks for your attention

Gianni Moretti