Hybrid ECS Scene-Graph Architecture optimized for Parallelization

Bălăban Iosua-Patrik

I. Modeling the Experimental Component

1. Introduction

The purpose of this research is to investigate the performance of three different models for organizing entities in a game engine:

1. BASIC ECS – a classical Entity Component System with contiguous memory storage (arrays), optimized for cache-friendly access.

2. GRAPH TREE – a traditional hierarchical scene graph where nodes are connected through pointers, resulting in non-contiguous access patterns and high cache miss rates.

3. HYBRID ECS GRAPH TREE – the proposed model, which keeps the hierarchical scene graph but stores the entities inside each graph node in local contiguous arrays. The goal is to preserve hierarchical organization while reducing cache overhead.

The proposed hybrid system attempts to combine the flexibility of a scene graph with the performance advantages of a classical ECS.

2. Modeling the Data and the Experiments

The experimental data consists of:
1. The total number of entities stored in each model.
2. Execution time (in microseconds) for performing identical work operations:
    1. Iterating through all entities,
    2. Executing the same “work“ function on each,
    3. Traversing the internal structure of each model.

The initial experimental setup uses 131,072 entities (2^17), large enough to reveal cache-related performance characteristics.

For each model (ECS, Graph Tree, Hybrid), the following steps are executed:
1. Build the structure with the fixed number of entities.
2. Record STATE BEFORE WORK (initial timestamp).
3. Perform a full traversal and processing step over all entities (WORK).
4. Record STATE AFTER WORK (final timestamp).
5. Compute DELTA, representing the time required for processing.

The collected measurements:

GRAPH TREE STATE BEFORE WORK 0
GRAPH TREE STATE AFTER WORK 131074
GRAPH TREE DELTA 23215 μs

BASIC ECS STATE BEFORE WORK 0
BASIC ECS STATE AFTER WORK 131072
BASIC ECS DELTA 1389 μs

HYBRID STATE BEFORE WORK 0
HYBRID STATE AFTER WORK 131072
HYBRID DELTA 1487 μs

BASIC ECS demonstrates the best performance thanks to contiguous memory access and excellent cache locality.

GRAPH TREE is significantly slower due to:
1. pointer chasing,
2. memory fragmentation,
3. unpredictable access patterns for the CPU cache.

HYBRID performs very close to ECS:
1. work is done on contiguous arrays,
2. the hierarchy overhead is minimized,
3. traversal stays cheap because nodes contain clustered data.

These results strongly suggest that the proposed hybrid model offers real performance benefits.

3. Mathematical Model and Validation Criteria

Mathematical Model for BASIC ECS
In a traditional ECS:
1. Components of the same type are stored in contiguous arrays.
2. Iteration is linear: the CPU walks through memory sequentially.

This produces extremely predictable access patterns.
We want a formula that captures:
1. cost of accessing each entity,
2. CPU cache behavior,
3. memory bandwidth effects.

A simple model is:

$$\mathbf{T_{ECS}(N) = N \cdot C_{seq}}$$

Where:

N = number of entities

C_{seq} = amortized cost of sequential access (includes: L1 hits + prefetching + CPU pipeline efficiency)

Sequential access has:
1. low miss rate often <5%
2. prefetcher automatically fetches next cache lines
3. pipeline rarely stalls

Thus Cseq is very small and nearly constant.

Mathematical Model for SCENE GRAPH
The scene graph is a pointer-based tree structure.

Each node may be anywhere in memory.
For each node:

1. Access its data
2. Then follow pointers to children
3. Then follow pointers to siblings

Each pointer dereference is unpredictable.

$$\mathbf{T_{graph}(N) = \sum_{i=1}^{N} (C_{ptr}(i) + C_{miss}(i))}$$

Where:

$C_{ptr}(i)$ = cost of following a pointer (loading the address from memory)
$C_{miss}(i)$ = cost of cache misses caused by pointer chasing

Tree traversals suffer from frequent cache misses because nodes are typically stored in non-contiguous memory. As a result, many pointer dereferences fetch data from main memory rather than the CPU cache. Since main-memory access is orders of magnitude slower, traversal time becomes dominated by these memory stalls. Thus, in pointer-based structures like scene graphs, pointer chasing is the primary performance bottleneck.

Mathematical Model for HYBRID
Hybrid stores:

1. the hierarchical tree structure, same as scene graph,
2. but each node stores a contiguous cluster of entities.

Thus two costs matter:

1. Traversing the tree (pointer chasing, small cost)
2. Processing entity arrays inside each node (sequential, fast)

Tree Navigation Cost
Each frame, the system must walk the scene-graph hierarchy:

1. Pointer jumps between nodes
2. Possible cache misses
3. Similar (but smaller) overhead to a pure graph tree

We represent this as:

TreeCost = cost of traversing parent - children links

This is the slower part.

Chunk Iteration Cost
Inside each node, components/entities are stored in compact arrays, so iterating them is fast:

1. Very few cache misses
2. Good spatial locality
3. Similar to a classic ECS

We represent this as:

ChunkCost = cost of iterating contiguous arrays inside each node

This is the faster part.

The full traversal cost is:

$$T_{hybrid}(N,k) = TreeCost(k) + ChunkCost(N)$$

Where:

1. N = total number of entities
2. k = number of graph nodes
3. TreeCost(k): overhead from the graph itself (pointer jumps)
4. ChunkCost(N): work done inside ECS-style arrays (fast loops)

Because each node holds entities in contiguous memory, most work happens inside fast ECS chunks. The slower pointer-chasing part only happens when switching nodes, not for every entity. Thus the hybrid system achieves:

1. Nearly ECS-level performance for iteration
2. With scene-graph hierarchy flexibility

Validation Strategy

Validation involves:

1. Running the experiment multiple times and checking consistency.
2. Testing across various scales (e.g., 2^10 to 2^20 entities).
3. Comparing against theoretical cache behavior from CPU documentation.

Comparing with established ECS performance benchmarks in academic and industry literature.

II. Case Study on Initial Data

1. Methodology

In this case study, we implemented:

1. a classical ECS,
2. a traditional scene graph,
3. the proposed hybrid ECS-hierarchical structure.

The implementation is done in Zig to ensure precise memory control and accurate performance measurement.

1.1. Technology Motivation

Zig provides low-level control over memory, perfect for ECS and cache experiments.
Alternative - Rust: safer but introduces constraints due to borrow checker complexity.
Alternative - C# DOTS/Unity ECS: extremely high performance but relies on a large runtime and may interfere with controlled experiments.

1.2. Data Generation

All structures are populated automatically with identical entities to eliminate structural bias and ensure fair comparison.

2. Results on the Initial Dataset

2.1. Observed Performance

1. The scene graph is roughly 20 times slower than ECS.
2. ECS and Hybrid implementations perform nearly identically.
3. Hybrid retains the hierarchical benefits without paying the pointer-chasing penalty.

Behavior aligns with established ECS literature regarding cache locality.
Observed performance correlates with CPU architectural expectations:

1. sequential access leads to highly cache efficient,
2. pointer-chasing leads to expensive and slow.

III. Related Work

1. Existing Approaches in the Literature

1.1. Popular ECS frameworks

1. EnTT
2. Flecs
3. Unity DOTS / ECS
4. Bevy ECS

These systems use SOA or AOSOA layouts to maximize cache performance.

1.2. Scene Graphs in Modern Game Engines

1. Godot (Node tree)
2. Unreal Engine (Actor + SceneComponent hierarchy)
3. Unity (GameObject hierarchy)

Common issues:

1. fragmented memory,
2. heavy pointer usage,
3. poor cache locality.

1.3. Comparison with the Proposed Approach

Feature	**ECS**	**Scene Graph**	**Proposed Hybrid**
Cache locality	excellent	poor	very good
Hierarchical structure	no	yes	yes
Pointer overhead	minimal	high	low
Scalability	excellent	moderate	excellent

The proposed hybrid preserves the benefits of ECS while supporting natural hierarchical scene structure, which pure ECS systems cannot easily provide.