UNIVERSITÀ
DEGLI STUDI
FIRENZE

Pattern Recognition with OpenMP

Parallel Computing

Alessandro Bianco 7147240

A.A. 2025/2026

Introduction

In an increasingly information-driven world, time series pattern recognition has become an essential tool for uncovering hidden trends, forecasting future events, and supporting data-driven decision-making.

Key applications of this approach span critical domains:

- Healthcare
- Social sciences
- Finance and economics
- …

Problem Definition

The problem of pattern recognition applied to time series consists in identifying, within a set of temporal sequences, the one that most closely resembles a reference sequence, known as the **query**. Formally:

$$\mathrm{S} = \{\mathrm{s}_1, \mathrm{s}_2, \dots, \mathrm{s}_\mathrm{N}\}$$
$$\mathrm{Q} = \{\mathrm{q}_1, \mathrm{q}_2, \dots, \mathrm{q}_m\}$$

where S is the large time series of length N and Q is the reference sequence (i.e the query) of length $\mathrm{m} \leq \mathrm{N}$

The Goal

The goal is to find the subsequence of S that minimizes a distance metric **SAD(S,Q)** with respect to Q. This distance (**sum of absolute differences**) can be expressed as:

$$SAD(S, Q) = \sum_{i=1}^{n} |s_i - q_i|$$

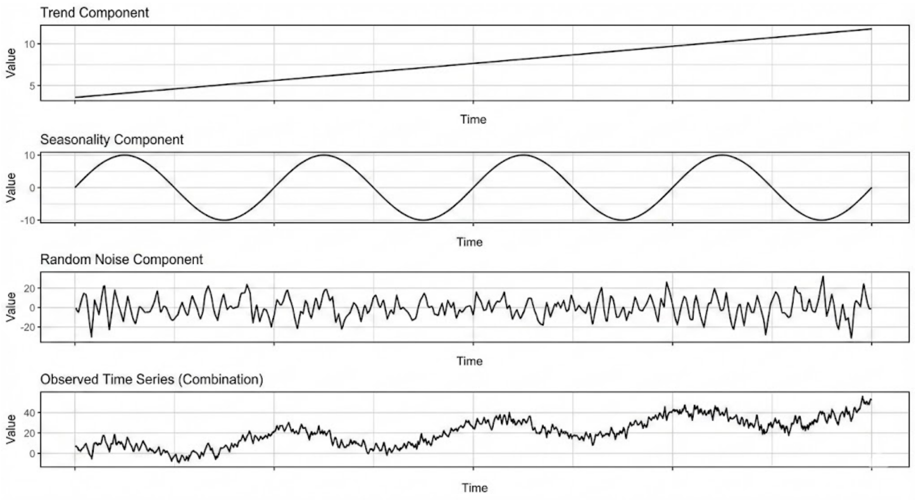
where the **lower** the SAD value, the greater the **similarity** between the analyzed series and the query.

Dataset Generation

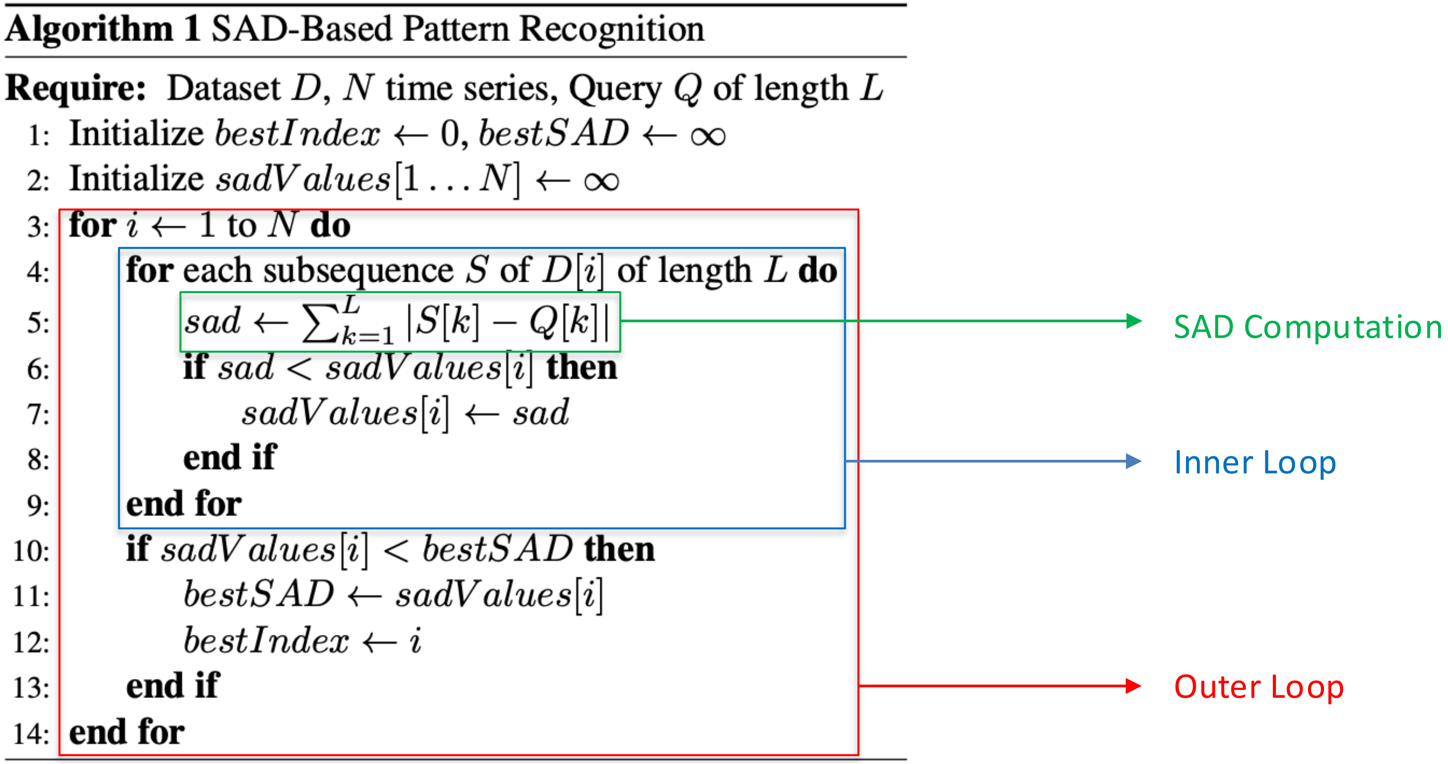
The dataset used in this project is generated using the **mockseries** Python library. A script is implemented to take as input the **number of time series** to generate, the **length of each series**, and the **length of the query**.

The time series generation algorithm creates synthetic sequences by combining three components:

- **Linear trend**
- **Stagionalty**
- **Random Gaussian noise**



Sequential Implementation



Algorithm 1 SAD-Based Pattern Recognition

```
Require: Dataset $D$, $N$ time series, Query $Q$ of length $L$
Initialize $bestIndex \leftarrow 0$, $bestSAD \leftarrow \infty$
Initialize $sadValues[1 \dots N] \leftarrow \infty$
for $i \leftarrow 1$ to $N$ do
    for each subsequence $S$ of $D[i]$ of length $L$ do
        $sad \leftarrow \sum_{k=1}^{L} |S[k] - Q[k]|$
        if $sad < sadValues[i]$ then
            $sadValues[i] \leftarrow sad$
        end if
    end for
    if $sadValues[i] < bestSAD$ then
        $bestSAD \leftarrow sadValues[i]$
        $bestIndex \leftarrow i$
    end if
end for
```

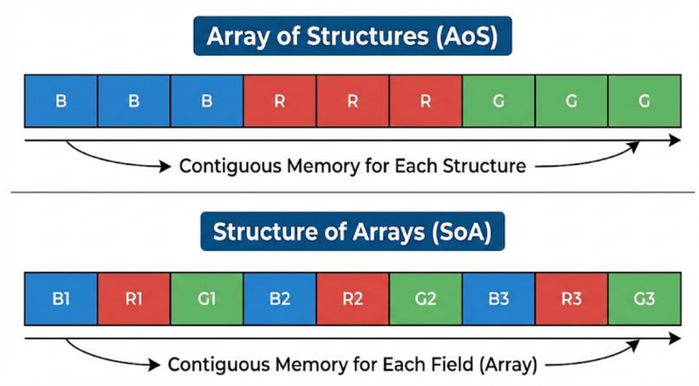
Memory layout: SoA vs AoS

SoA: Structure of Arrays

- The values of the series are stored in a vector of doubles, indexed by instants of time.
- Each instant of time contains the values of all the series, stored contiguously.
- Operations on multiple series at the same instant read contiguous data, but penalizes sequential access due to memory jumps.

AoS: Array of Structures

- Data stored via the *Sample* data structure which represents the single series.
- Samples from the same series are contiguous in memory, optimizing sequential access.
- Structure less suitable for parallel computation


Array of Structures (AoS)
B
B
B
R
R
R
G
G
G
Contiguous Memory for Each Structure
Structure of Arrays (SoA)
B1
R1
G1
B2
R2
G2
B3
R3
G3
Contiguous Memory for Each Field (Array)

Parallelization strategies

Two parallelized variants will be developed for both the *AoS* and *SoA* implementations. The aim is to investigate how **parallelizing two different loops** within the same algorithm can **affect performance** and determine the most suitable usage scenarios.

Outer Loop Parallelization

Inter-series

- Iterates over the various time series in the dataset
- Here we assign time series to different and separated threads

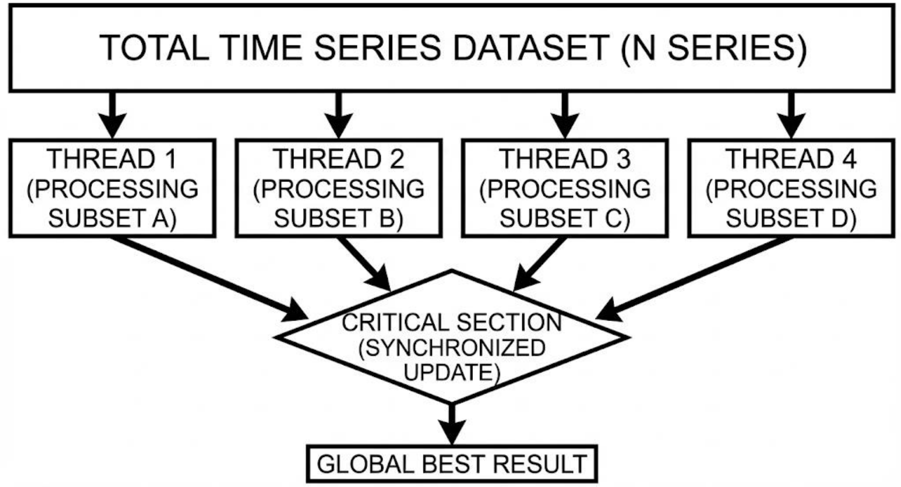
Inner Loop Parallelization

Intra-series

- Scans each individual time series to compare it against the query
- We maintain the sequential loop over the series but we parallelize the internal comparison between the series and the query

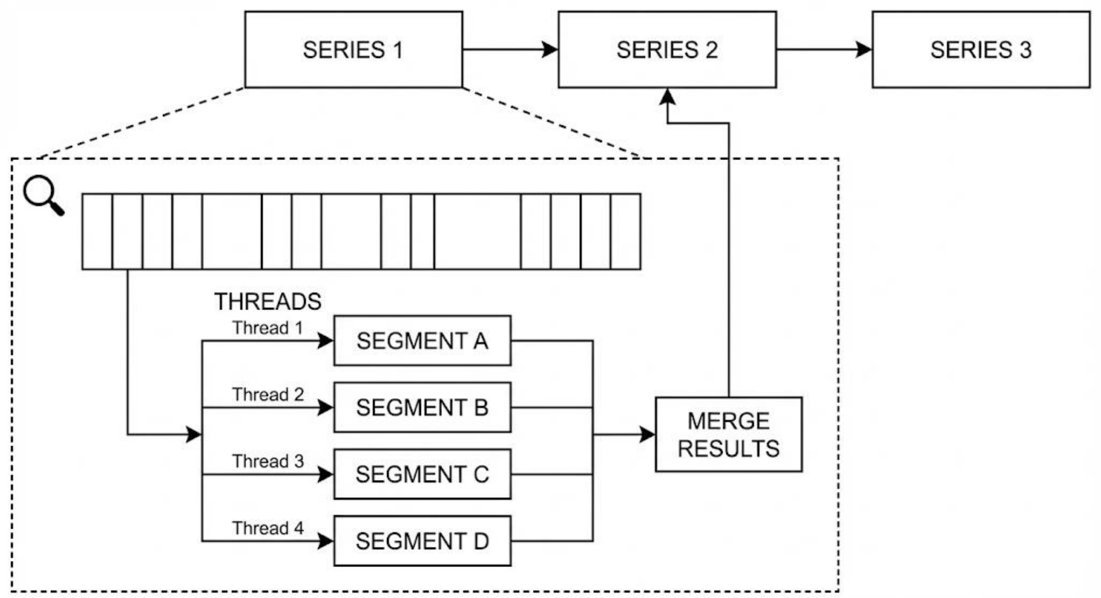
Outer Loop Parallelization

- *#pragma omp parallel for* is applied to the outer loop for the time series iteration
- *bestSad* value is efficiently managed via the *reduction* clause
- *bestIndex* updates require a *#pragma omp critical* section to prevent race conditions


TOTAL TIME SERIES DATASET (N SERIES)
THREAD 1 (PROCESSING SUBSET A)
THREAD 2 (PROCESSING SUBSET B)
THREAD 3 (PROCESSING SUBSET C)
THREAD 4 (PROCESSING SUBSET D)
CRITICAL SECTION (SYNCHRONIZED UPDATE)
GLOBAL BEST RESULT

Inner Loop Parallelization

- Outer loop remains sequential, and parallelism is applied strictly to the inner loop
- For each time series a team of threads is spawned to compute SAD across all window positions
- The global minimum is upddated sequentially by the main thread, eliminating the need for critical sections



Testing the implementations: Test suite

The tests to be carried out aim to determine which data structure can offer better performance for each parallelization strategy, and how the size of the datasets to be analyzed may affect the effectiveness of the strategies (Inner vs. Outer).

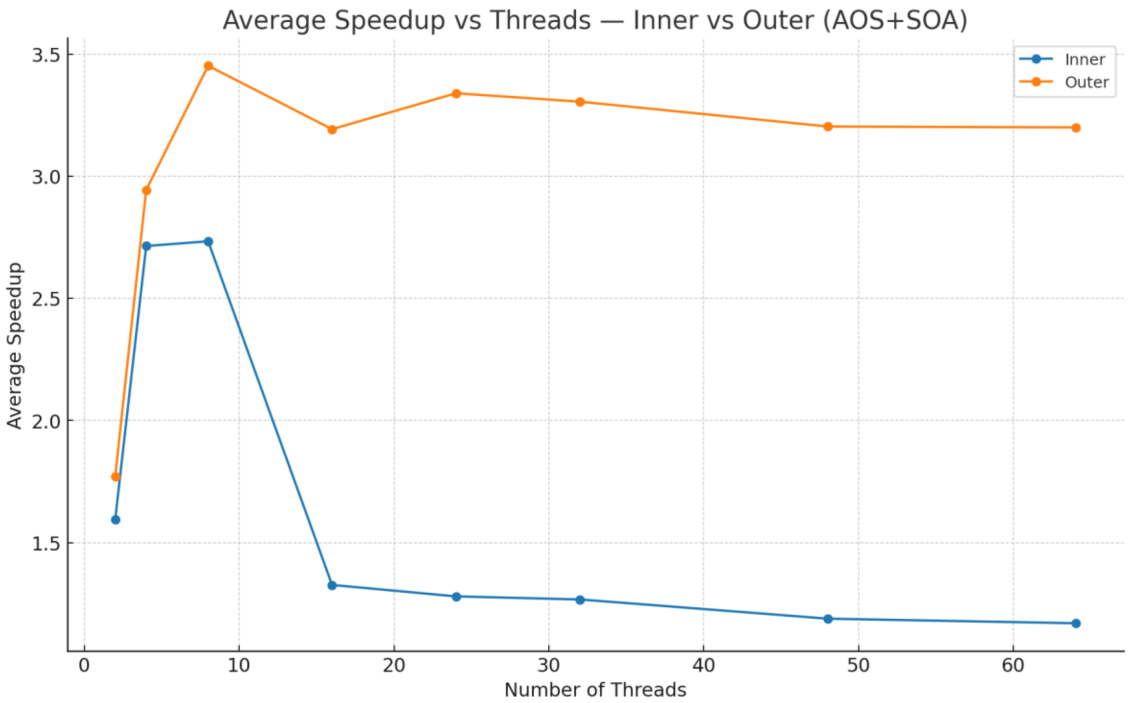
- **Outer Parallelization Focused**: Verify the effectivness of outer parallelization when there are many series to process in parallel, with reduced synchronization overhead

# Series	Series Length	Query Length
1000	100 points	50 points
5000	100 points	50 points

- **Inner Parallelization Focused**: Test inner parallelization on very long series, where parallelism over the time steps should become more advantageous

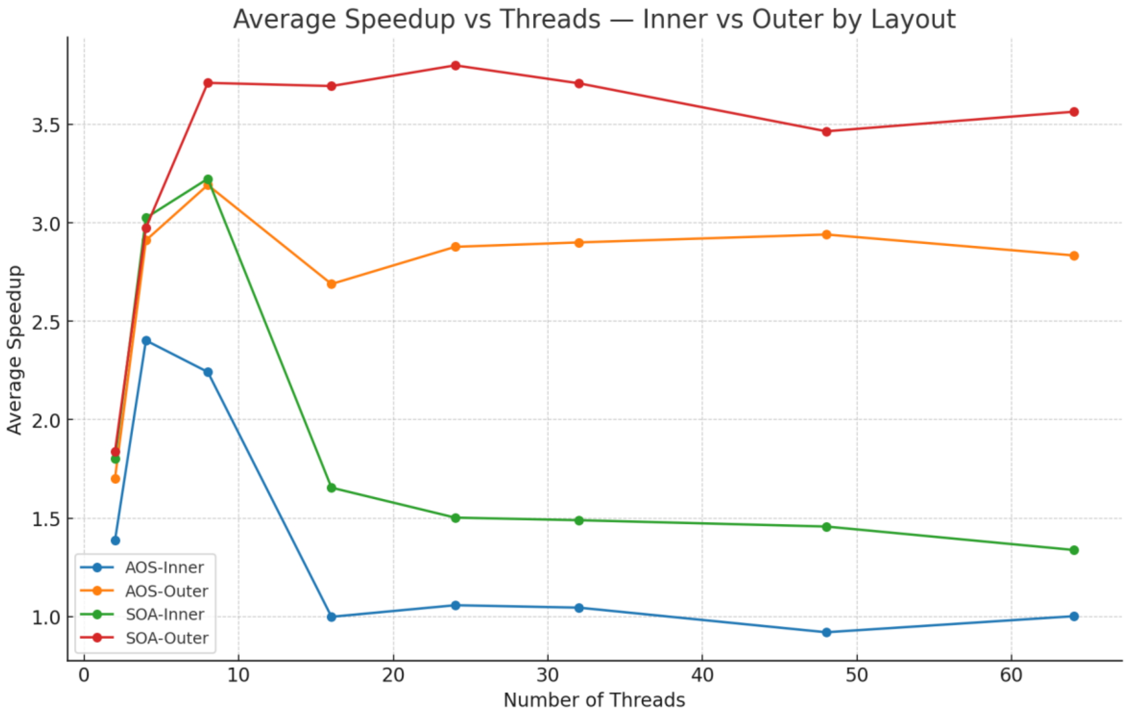
# Series	Series Length	Query Length
10	5000 points	50 points
5	10000 points	50 points

Test Results: Inner vs Outer



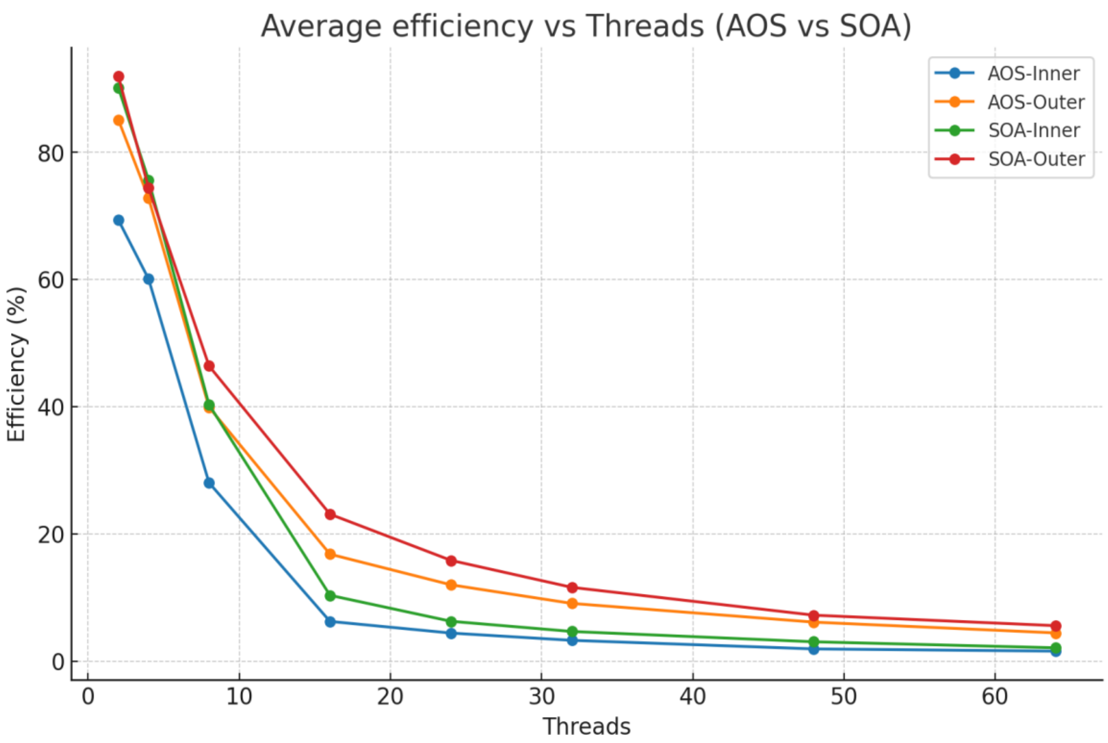
- Outer-loop parallelization delivers the highest speedup due to independent workloads and low overhead.
- Inner-loop parallelization degrades beyond 8 threads because of overhead and false sharing.
- Both approaches hit a performance plateau caused by memory bandwidth saturation.

Test Results: Memory Layouts



- SoA is the most efficient layout for parallel computation, but SoA–Inner degrades rapidly.
- AoS–Outer, despite a less optimal memory layout, maintains better performance at high thread counts.
- The benefits of SoA’s contiguous memory accesses are canceled by the synchronization and thread-management overhead of inner parallelization.
- The coarse-grained nature of outer parallelization compensates for the scattered memory accesses of AoS.

Test Results: Efficency



- All curves show a rapid decline as the number of threads increases, in accordance with Amdahl's Law.
- As parallelism grows, synchronization and thread-management overhead becomes dominant over useful computation time.
- The Outer configuration maintains higher efficiency thanks to its embarrassingly parallel nature.
- The Inner configuration degrades more rapidly as thread count increases.

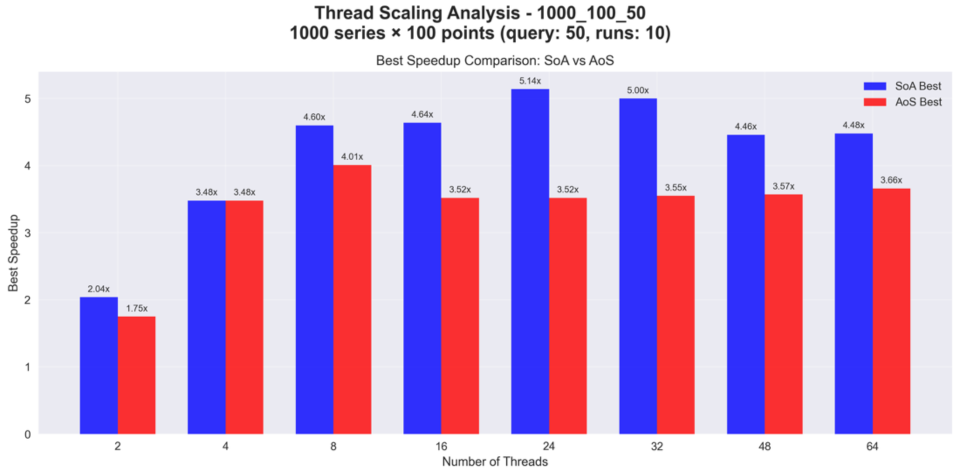
Conclusions

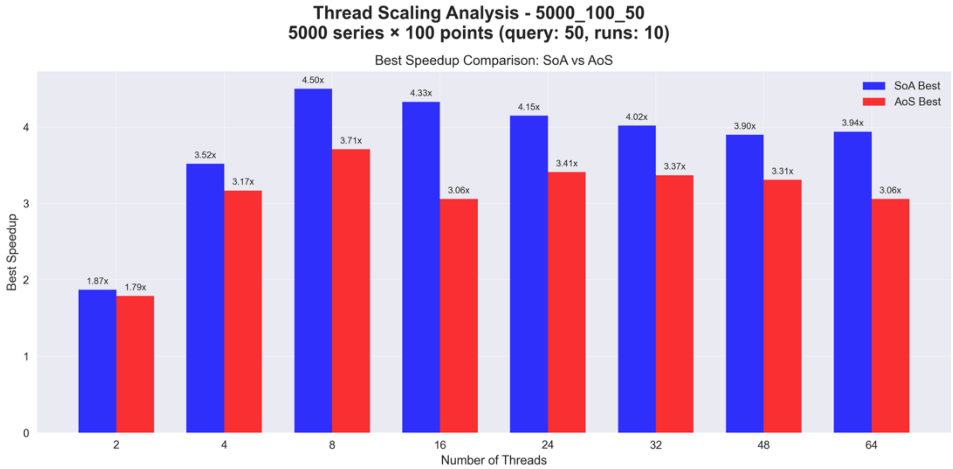
- **Strategy**: *Outer Dominates Inner*
  Outer Parallelization scales best by minimizing synchronization, while the inner strategy suffers from high overhead (barriers, false sharing) and poor data locality.

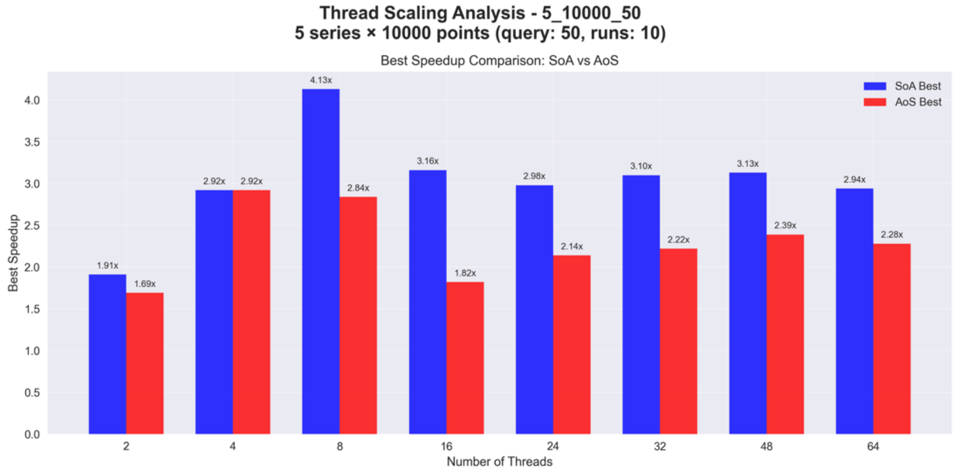
- **Layout**: *SoA Outperforms AoS*
  SoA is systematically more efficient in parallel due to contiguous, cache-friendly memory access while AoS remains competitive mostly in sequential scenarios.

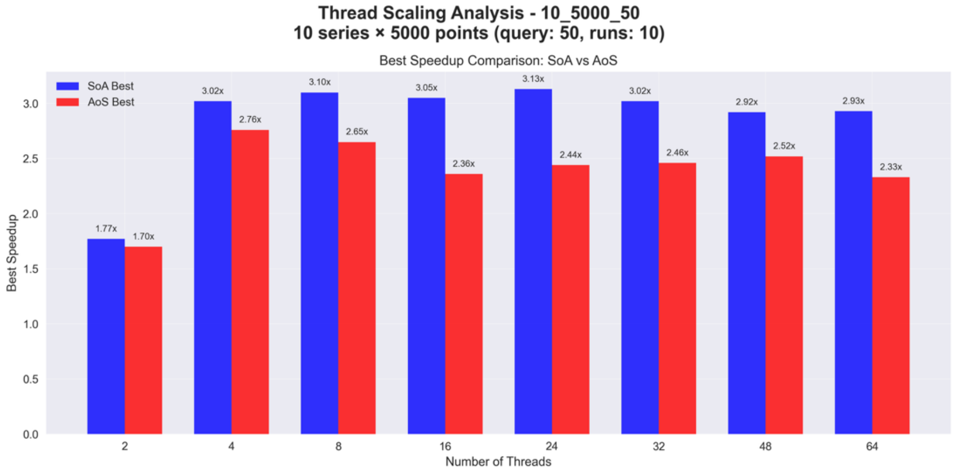
- **Hardware Limits**:
  Performance plateaus beyond 8–16 threads due to bandwidth saturation and efficiency drops monotonically, consistent with Amdahl's Law.

Test Results


Thread Scaling Analysis - 1000_100_50
1000 series × 100 points (query: 50, runs: 10)
Best Speedup Comparison: SoA vs AoS
SoA Best
AoS Best
Best Speedup
Number of Threads
2.04x
1.75x
3.48x
3.48x
4.60x
4.01x
4.64x
3.52x
5.14x
3.52x
5.00x
3.55x
4.46x
3.57x
4.48x
3.66x


Thread Scaling Analysis - 5000_100_50
5000 series × 100 points (query: 50, runs: 10)
Best Speedup Comparison: SoA vs AoS
SoA Best
AoS Best
Best Speedup
Number of Threads
1.87x
1.79x
3.52x
3.17x
4.50x
3.71x
4.33x
3.06x
4.15x
3.41x
4.02x
3.37x
3.90x
3.31x
3.94x
3.06x


Thread Scaling Analysis - 5_10000_50
5 series × 10000 points (query: 50, runs: 10)
Best Speedup Comparison: SoA vs AoS
SoA Best
AoS Best
Best Speedup
Number of Threads
1.91x
1.69x
2.92x
2.92x
4.13x
2.84x
3.16x
1.82x
2.98x
2.14x
3.10x
2.22x
3.13x
2.39x
2.94x
2.28x


Thread Scaling Analysis - 10_5000_50
10 series × 5000 points (query: 50, runs: 10)
Best Speedup Comparison: SoA vs AoS
SoA Best
AoS Best
Best Speedup
Number of Threads
1.77x
1.70x
3.02x
2.76x
3.10x
2.65x
3.05x
2.36x
3.13x
2.44x
3.02x
2.46x
2.92x
2.52x
2.93x
2.33x