# Performance Analysis of Parallel K-Means

Color Quantization Optimization using Numba & Multi-Core CPU

Pere Llauradó Adeva | Final Project

# Problem Introduction

# CONTEXT & MOTIVATION

## Image Colors Quantization

- Reduce the number of colors while preserving visual fidelity.
- Essential for image compression.
- Fastest processing utile for example in medical or biological images

# COMPUTATIONAL CHALLENGES

### Complexity

Up to 6.0x10^9 operations for 1024x1024 pixels images uting 32 clusters. This leads to massive CPU load during distance calculations.

### The GIL & Interpreter

Python's Global Interpreter Lock prevents real multithreading. Standard loops cannot use multiple cores effectively.

### Race Conditions

Centroid updates risk data corruption if multiple threads write to the same memory address simultaneously.

# The Approach

# K-MEANS ALGORITHM

## 1. Initialization (Pre-loop)

- Converts the image  into a long list of pixels
- Select K random pixels from the original img. to act as the initial color centers

## 2. Update Phase

**A- Assignment:**

Calculate distance for every pixel to each of the centroids and assign a label to the pixel

**B- Update:**

Calculate new centroids by taking the mean of the color values of all pixels assigned to that cluster

**C- Convergence:**

- How much the centroids have moved
- If the movement is less than the tolerance the algorithm stops early.

## 3. Final Reconstruction

- Once the centroids are finally determined
- Assign to each pixel the colour of its centroid

# PARALLELIZED PARTS

## 2.A. Assignment (Prange)

- Parallel pixel distribution via Numba prange.
- Each thread processes a chunk of pixels simultaneously

## 2.B. Update Phase (Manual)

- Manual chunking to manage thread-local data.
- Each thread calculates its own partial sums

## 2.C. Reconstruction

- Parallel pixel distribution via Numba prange.
- Each thread processes a chunk of pixels simultaneously

# DATASETS USED

## Dataset 1 — Landscape Pictures:

High resolution, RGB pictures with resolutions of: 1024x768, 1024x1024, 1600x319… pixels

## Dataset 2 — Intel Image Classification:

Low resolution, RGB pictures with a resolution of 150x150 pixels

# K-MEANS VISUAL RESULTS

## Dataset 1 — Landscape Pictures:

K = 8

K = 16

K = 32

# K-MEANS VISUAL RESULTS

## Dataset 2 — Intel Image Classification:

K = 8

K = 16

K = 32

# DATASETS & WORKLOAD COMPLEXITY

| Dataset Name | Resolution | Pixels (N) | Ops (K=32) |
|---|---|---|---|
| **Small_Res** (Intel Classification) | 150 x 150 | 22,500 | ~1.3 x 108 |
| **Full_Res** (Landscape Pictures) | 1024 x 1024 | 1,048,576 | ~6.0 x 109 |

# PARALLELIZATION STRATEGY

## Native Speed Execution

By using Numba's @jit(parallel=True), we bypass the GIL entirely and compile the algorithm into LLVM machine code.

This allows the 8 logical threads of our Core i7 to saturate memory bandwidth and maximize floating-point throughput.

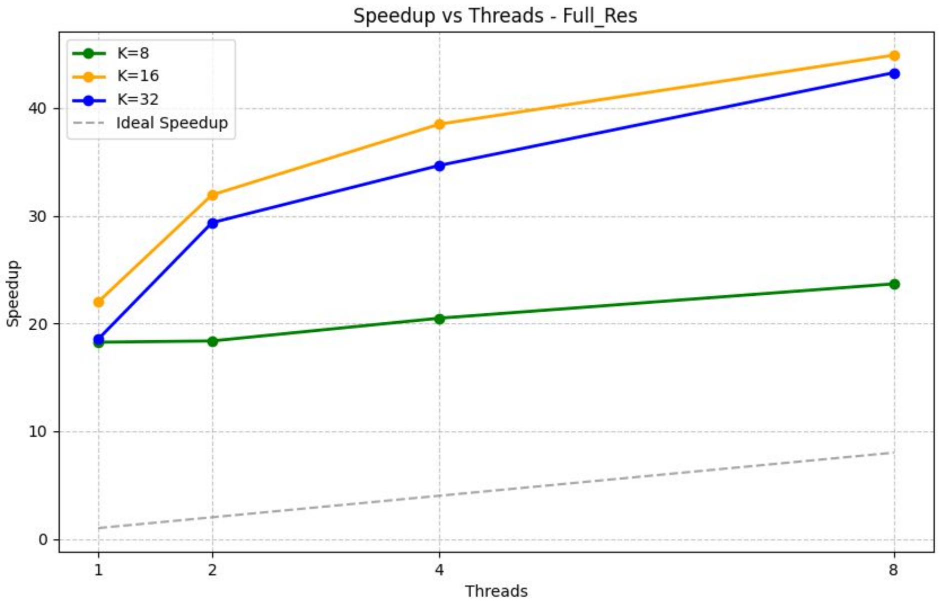
Automatic and manual work distribution: using orange when the operation is fully independent.  In the risky cases we use manual distribution.

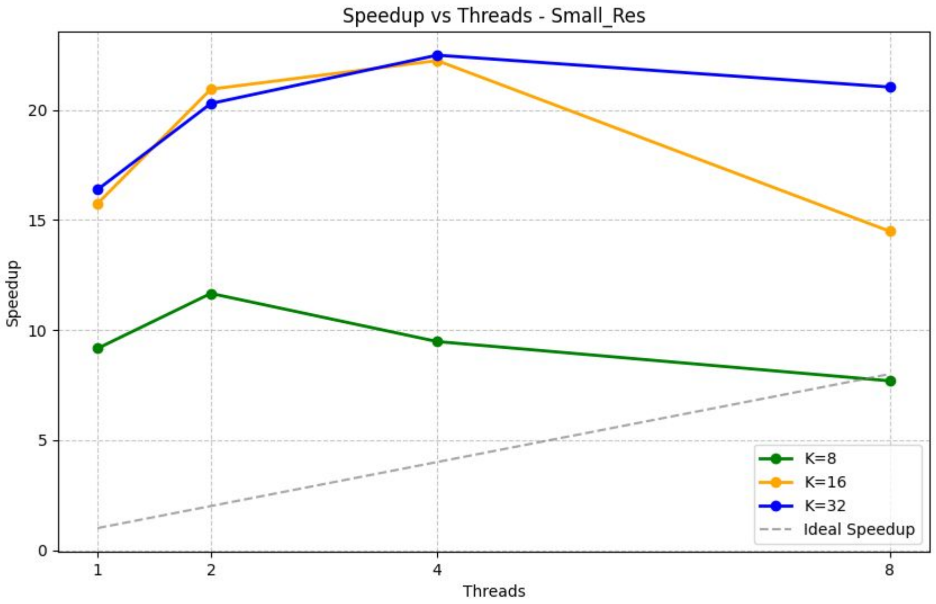
# KEY OPTIMIZATIONS

- **False Sharing Prevention:** Added 64-byte padding to private thread buffers, ensuring data stays on separate L1 cache lines.
- **JIT Warm-up:** Pre-compiling functions before measurement to exclude overhead from LLVM compilation time.
- **Array Flattening:** Reshaping 3D images to 2D contiguous arrays for optimal CPU cache prefetching.
- **Inertia Validation:** Comparing Sum of Squared Errors rather than exact colors to calculate the accuracy of the algorithm

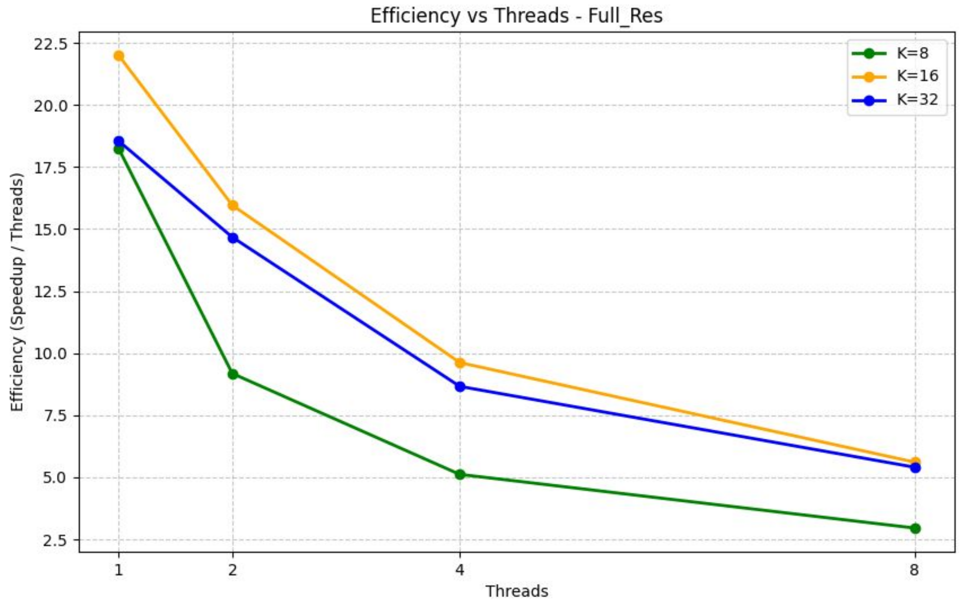
# Performance Results

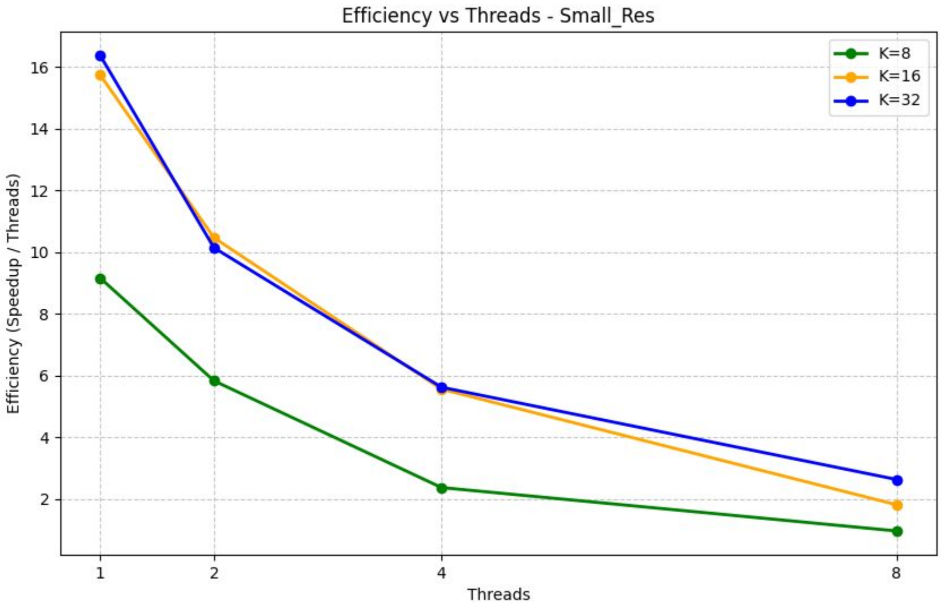
# SPEEDUP VS THREADS



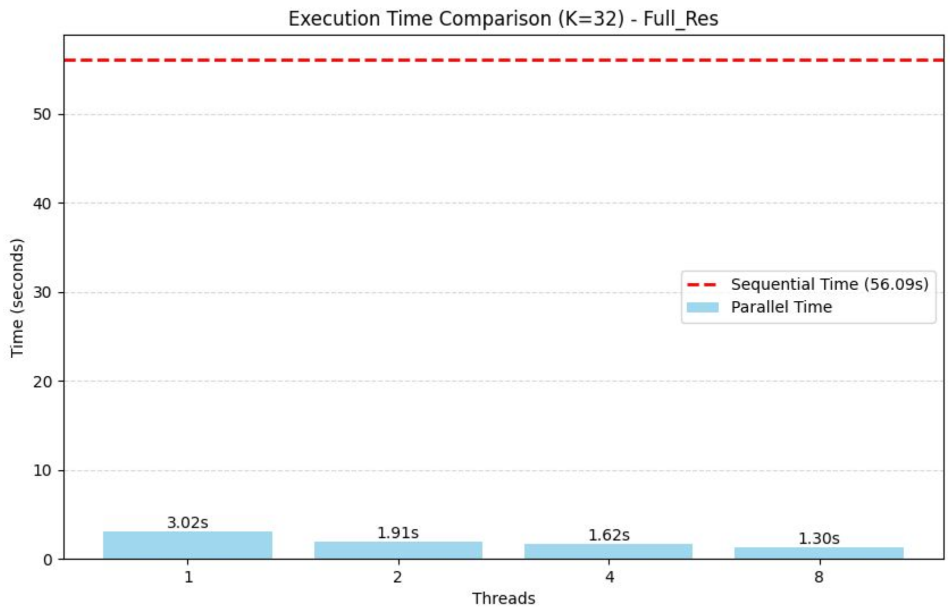



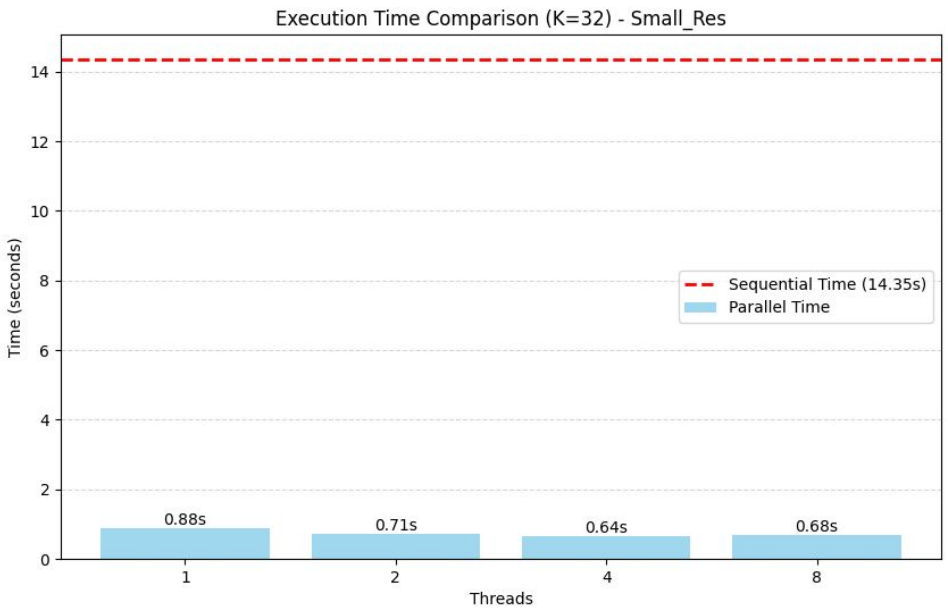
# EFFICIENCY VS THREADS



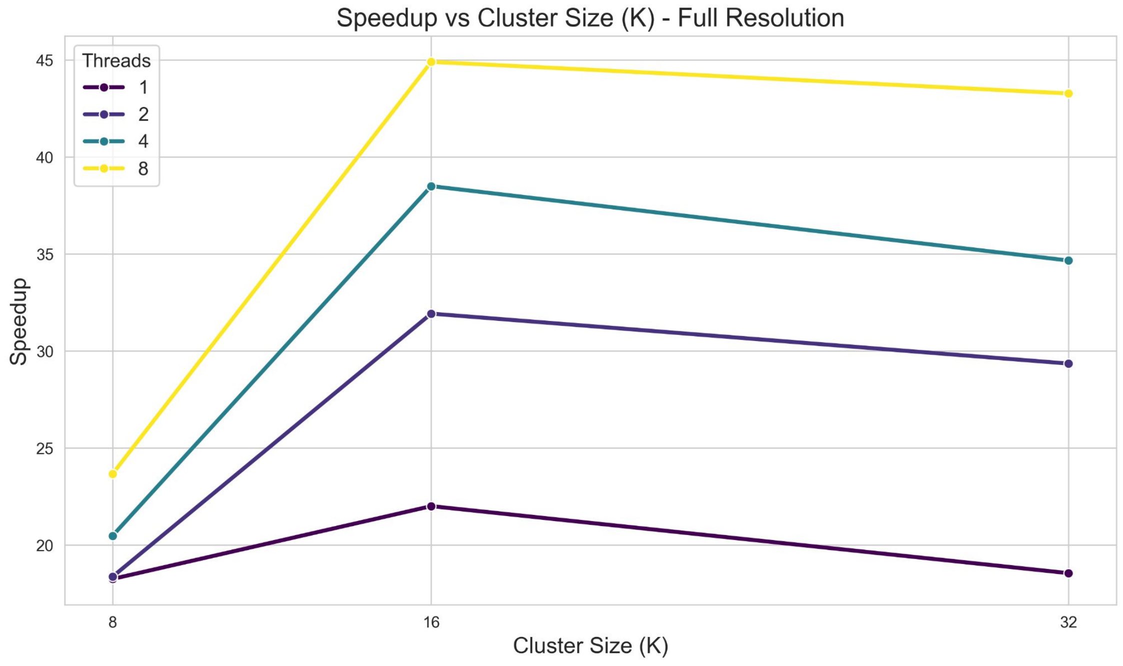



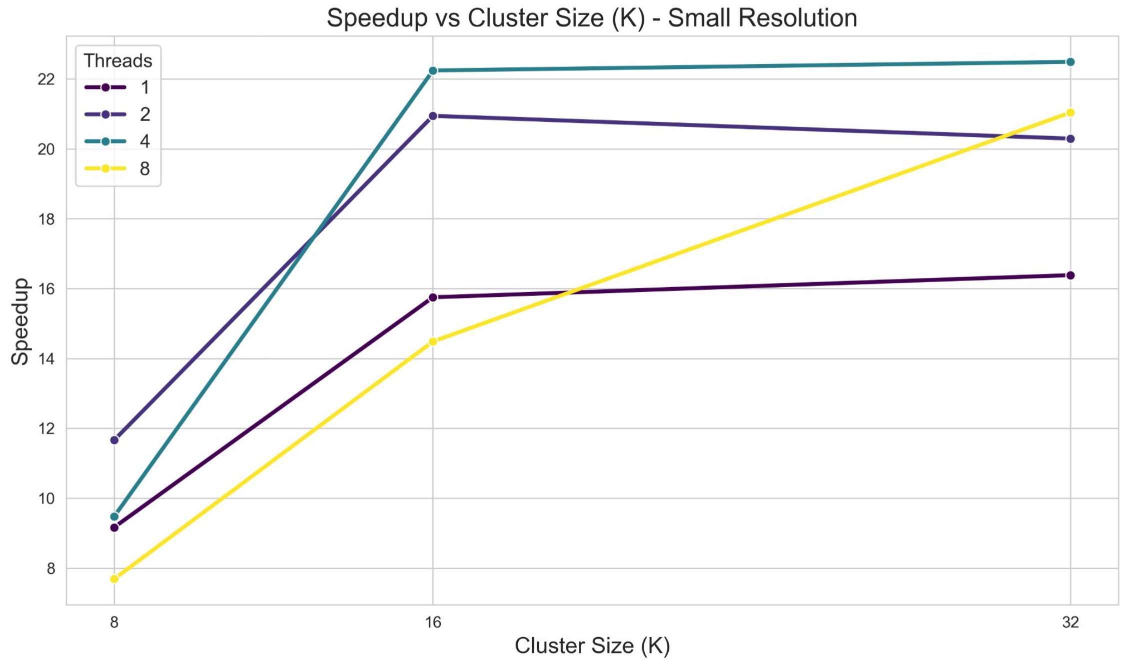
# TIME VS THREADS



# SPEEDUP VS CLUSTER SIZE



# BEST CONFIGURATIONS

**Hw settings:** i7-8550U with 4 physical cores and 8 logical processors utilizing Hyper-Threading

**Threads:** 8 is the best number of threads for Dataset 1 and 4 for Dataset 2

**Clusters:** for the Dataset 1 is K = 16 and for the Dataset 2 is K = 32

## Dataset 1: High resolution

## 44.9x Speedup

## Dataset 2: Low resolution

## 22.49x SpeedUp

# INTERESTING FINDINGS OR SURPRISES

## Validation problem

- Floating-Point sum error: float limited size make microscopic rounding differences
- The "Butterfly Effect" in K-Means: A tiny rounding error can flip a single pixel to a different cluster.
- Different correct solutions
- Total error comparison

## Empty clusters

- On the small dataset, when we tested with K = 32, some clusters ends up being orphaned, without any pixel belonging to it.
- **First Idea:** pick a random pixel as the new centroid
- **Correct idea:** leave the centroid as black [0,0,0] == deleting it, to avoid parallelization problems

# Conclusions & Lessons

# SUMMARY

## Performance gains

- Reduce the processing time from 56 s to 1.29 s with a 44,9x speedup.
- Numba can pass the python GIL to accelerate and parallelize heavy image processing

## Optimal configuration

- **High resolution dataset:** 8 threads an K = 16
- **Low resolution dataset:** 4 threads and K = 32

## Scalability

- **Full_Res images:** scale significantly better (Gustafson's Law) as the computational load hides thread management overhead.
- **Small_Res images:** are limited by Amdahl's Law, where fixed synchronization costs dominate the short execution time.

# LESSONS LEARNED

- **Power of compilation:** eliminating the python interpreter reduces more the execution time than using threads
- **Cache sizes:** understand the size of the cache and add the necessary padding to avoid threads fighting for memory access.
- **Validation correctness importance:** depending on the algorithm and the size of the problem the result of the same execution could be slightly different. Important to consider it for the tests.

# Thank You!

Questions or Discussion

# Bibliografy


- [1] Dataset 1 (Big images):Kaggle - Landscape Pictures
- https://www.kaggle.com/datasets/arnaud58/landscape-pictures?resource=download
- [2] Dataset 2 (Small images):Kaggle - Intel Image Classification
- https://www.kaggle.com/datasets/puneet6060/intel-image-classification?resource=download
- [3] Parallel Programming Course Lecture Notes and Slides. Course material from the Parallel Programming course.