Rough Draft:

Introduction

With the rise of artificial intelligence, image processing has become increasingly important for applications such as object detection, image segmentation, and classification. As AI models grow in complexity, the computational demands for processing images efficiently also increase. To address these challenges, multithreaded processing is commonly used to accelerate computations. However, traditional methods, such as OpenCV's stripe-based image partitioning, may not always be optimal for load balancing and efficiency.

This project explores an alternative multithreading approach, where images are recursively divided into smaller sections using a 4^N partitioning method. The goal is to analyze whether this approach can provide better workload distribution and improved computational efficiency compared to OpenCV's default method. Through experimentation and performance evaluation, this research aims to determine whether a more effective multithreading strategy can be developed for high-performance image processing tasks.

Background

What is OpenCV?

OpenCV is an open-source computer vision library designed for image processing, object detection, classification, and machine learning applications. It provides a wide range of optimized algorithms for handling visual data efficiently. Originally written in C++, OpenCV also supports Python, Java, JavaScript, Rust, and other languages, making it widely accessible for various development environments.

Multithreading in OpenCV

OpenCV includes built-in multithreading capabilities to enhance performance in computationally intensive tasks. It achieves parallel processing by dividing an image into horizontal strips, where the height of the image is split among the available threads. Each thread is assigned to a rectangular section of the image, partitioned horizontally, and processes that portion independently.

Methodology

This project explores an alternative image partitioning method tailored for parallel processing. Instead of dividing the image into horizontal stripes like OpenCV, the proposed approach recursively partitions the image using 4^N. At the base level (N = 0), the entire image is treated as a single unit. As N increases, the image is subdivided into four smaller sections at each level, creating a hierarchical structure for processing.

- When N = 1, the image is split into 4 equal parts: top-left, top-right, bottom-left, and bottom-right.
- When N = 2, each of the 4 sections from N=1 is further divided into 4 smaller boxes, creating 16 total sections.
- This recursive division continues for higher values of N, increasing the number of sections exponentially at the rate of N^4.

This testing uses OpenCV version 4.11.0 to simulate both the striped-based method and the 4^N partitioning method. OpenCV has built-in multithreading support; however, it is disabled when using the 4^N partitioning method. The 4^N partitioning method is restricted to using 4^N threads, and will be tested at **N = 1**, **N = 2**, and **N = 3**. The horizontally partitioned method will be tested using 4, 16, and 64 threads. A base case of 1 thread will also be used. For testing, we will process 50 high complexity images and 50 low complexity images separately.

High complexity images are large and contain intricate details, requiring more computational resources to process. These images will demonstrate how the multi-threading method scales with an increased workload, revealing whether there are diminishing returns or overhead when using multiple threads.

Low complexity images are smaller and less detailed, making them easier to process. These images will help assess how the multi-threading methods perform when processing light workloads. This comparison will highlight potential overhead when minimal computation resources are required.

(metrics will be added later)

Results

There are no results yet, testing is still ongoing. Graphs will be used to show the difference in each metrics being used.

Conlcusion