# Analysis Report

h228zhou

## Project Description

This project involves the development of an image processing system designed to run on a CPU. The system reads a series of images and applies specified effects to them using image convolution techniques. To enhance computational efficiency and leverage concurrency, two parallel implementations are provided in addition to the sequential version:

1. **Parallel Files Pattern (Parfiles):** Processes multiple images in parallel, with each image being handled by a separate goroutine or thread.
2. **Parallel Effects Pattern (Parslices):** Processes individual images in parallel by dividing each image into slices (horizontal bands) and applying effects concurrently across these slices.

We conducted experiments to benchmark these implementations and generated speedup graphs to analyze their performance

## Instructions

To replicate the experiments and generate the speedup graphs, please follow these steps:

1. **Ensure Dependencies are Installed:**
   - **Go Programming Language:** Make sure Go is installed on your system.
   - **Python 3:** Required for running the benchmarking script.
   - **Matplotlib:** Install via `pip install matplotlib` for plotting graphs.
2. **Run the Benchmarking Script:**
   - `sbatch benchmark-proj1.sh`

## Analysis

### Hotspots and Bottlenecks in the Sequential Program

In the sequential implementation, the primary hotspots and bottlenecks are:

1. **I/O Operations:**
   - **Image Loading and Saving:** Reading large images from disk (`Load` function) and writing processed images back to disk (`Save` function) consume significant time, especially with high-resolution images.
   - **Impact:** These operations are blocking and contribute to increased total execution time.
2. **Computational Intensity of Convolution:**
   - **Nested Loops Over Pixels:** The convolution operation involves iterating over each pixel and applying a kernel, resulting in a computational complexity of O(N^2) for an image with N x N dimensions.

- **No Parallelism:** In the sequential version, all computations are done on a single thread, which limits the utilization of available CPU cores.

## Performance Comparison of Parallel Implementations

Between the two parallel implementations, the **Parallel Files Pattern (Parfiles)** performs better than the **Parallel Effects Pattern (Parslices)**.

**Reasons for Better Performance of ParFiles:**

1. **Improved Thread Utilization:**
   - **Concurrent Image Processing:** ParFiles processes multiple images simultaneously, keeping all threads active and reducing idle time.
   - **Workload Distribution:** Each thread works independently on a complete image, leading to better load balancing.
2. **Reduced Overhead:**
   - **Minimal Synchronization:** Since each image is processed independently, there is little need for synchronization between threads.
   - **Lower Communication Costs:** Threads do not need to share data or communicate frequently, reducing overhead.

**Challenges with Parslices:**

- **Synchronization Overhead:**
  - Processing slices in parallel requires careful handling of edge pixels, leading to additional synchronization or data copying.
  - Convolution operations depend on neighboring pixels, necessitating overlap between slices and increasing complexity.
- **Increased Overhead from Data Copying:**
  - **Swapping Buffers:** The `SwapBuffers()` function involves copying data between buffers, which adds overhead and impacts performance.

## Impact of Problem Size on Performance

**Problem Size Effects:**

- **Large Datasets:**
  - With larger images, the ratio of computation to overhead increases, making parallelization more effective.
  - The overhead of spawning goroutines becomes negligible compared to the total computation time.
- **Small Datasets:**
  - For smaller images, the overhead of managing threads and synchronization can outweigh the benefits of parallelization.
  - Threads may complete their tasks quickly and remain idle, leading to suboptimal utilization.

**Observation from Experiments:**

- **Consistent Performance in Parslices:**

    - The speedup remains significant even as the number of threads increases from 8 to 12, indicating effective amortization of overhead.
- **Variability in Mixed Datasets:**
    - The mixed dataset shows lower speedups due to varied task sizes, causing load imbalance where some threads finish earlier than others.

## Comparison with Amdahl's Law

**Expected Speedup:**

Amdahl's Law predicts the maximum speedup achievable by parallelizing a portion of the program:

$$\text{Speedup} = 1/[(1 - P) + P/N]$$

Where P is the parallelizable portion of the program and N is the number of threads

The observed speedups in the experiments are generally in line with the expected values computed using Amdahl's Law.

- **Factors Affecting Speedup:**
    - The I/O operations remain sequential and limit the overall speedup.
    - Synchronization and communication overhead reduce the effective parallelizable portion.

**Discrepancies:**

At higher thread counts, the speedup plateaus due to increased overhead and the fixed sequential portion. Potential Causes:

- Uneven distribution of work leads to idle threads.
- CPU limitations and context switching overhead can impact performance.

## Potential Areas for Performance Improvement

**Reducing Synchronization Overhead:**

Overlap I/O with computation by using asynchronous read/write operations to minimize idle time.

**Algorithmic Enhancements:**

Further parallelize computations within slices if the overhead can be managed.

**Hardware Utilization:**

Optimize memory access patterns to improve cache performance, reducing latency.

# Graphs and Data Interpretation

*Note: Please refer to the attached graphs for visual representation.*

## Speedup vs. Number of Threads

- **Parfiles Pattern:**

    - **High Speedup with Increased Threads:** Shows significant performance gains up to a certain point before plateauing.
    - **Better Performance with Large Datasets:** Larger images benefit more due to higher computation loads.
- **Parslices Pattern:**
    - **Consistent Scaling:** Maintains a steady increase in speedup with more threads but may exhibit diminishing returns at higher counts.
    - **Impact of Synchronization:** Overhead from managing slices affects scalability.

### Conclusion from Graphs

There's an optimal number of threads beyond which additional threads do not contribute to performance gains.The Parfiles pattern is generally more efficient due to lower overhead and better thread utilization. Effective load distribution is crucial, especially for mixed datasets with varying image sizes.

# Conclusion

The project successfully demonstrates the benefits of parallelization in image processing applications. By implementing two parallel patterns, we observed significant performance improvements over the sequential version. The Parfiles pattern outperforms the Parslices pattern due to reduced synchronization overhead and better thread utilization.