Parallel Image Processing: Performance Analysis of OpenMP and MPI Implementations

Vlasan Robert-Cosmin
Inteligent Software Robotics
West University of Timisoara
robert.vlasan02@e-uvt.ro

June 28, 2025

Abstract

In this project i will present the performance comparasen analysis of parallel image processing algorithms that were implemented using OpenMP, MPI and seqventual. I implemented 4 computaionally intensive algorithms :Gaussian blur, edge detection, histogram equalization and Otsu thresholding. The performance evaluation was conducted on 2 sets of test, one set contains images ranging from 512x512 ti 2048x2048 and the other set contains images from 1024x1024 to 8192x8192 using 4 cores/processes. esults demonstrate significant speedups with Gaussian blur achieving the best performance at 3.04× speedup for OpenMP and 2.40× speedup for MPI, while edge detection achieved 1.78× OpenMP and 2.14× MPI speedup, while highlightring the trade-offs between shared-memory and distributed-memory parallel programming approaches.

1 Introduction

Applications for image processing are perfect candidates for parallel computing optimization since they demand a significant amount of computational power. The performance characteristics of four basic image processing algorithms built utilizing two different parallel programming paradigms—MPI for distributed-memory architectures and OpenMP for shared-memory systems—are assessed in this work.

Implementing sequential baseline algorithms, creating parallel versions of OpenMP and MPI, performing thorough performance analysis, and assessing parallel efficiency and scalability characteristics are the main goals.

2 Methodology

2.1 Algorithm Selection

Four algorithms were selected based on their computational intensity and parallelization potential:

Gaussian Blur: Implements 5×5 convolution with Gaussian kernel ($\sigma = 1.0$), providing smoothing and noise reduction. Complexity: $O(n^2 \times k^2)$ where n is image dimension and k is kernel size.

Edge Detection: Utilizes Sobel operators for gradient computation in X and Y directions, calculating edge magnitude as $\sqrt{G_x^2 + G_y^2}$. Complexity: $O(n^2)$.

Histogram Equalization: Enhances image contrast through cumulative distribution function (CDF) normalization: $h(v) = \text{round}\left(\frac{\text{CDF}(v)-\text{CDF}_{\min}}{M\times N-\text{CDF}_{\min}} \times (L-1)\right)$, where $M \times N$ is image size and L is intensity levels.

Otsu Thresholding: Determines optimal binary threshold by maximizing between-class variance: $\sigma_B^2(T) = \omega_0(T)\omega_1(T)[\mu_0(T) - \mu_1(T)]^2$, where ω represents class probabilities and μ represents class means.

2.2 Parallel Implementation Strategies

OpenMP Implementation: Utilized loop-level parallelization with `#pragma omp parallel for` directives. Key optimizations included:

- `collapse(2)` for nested loop parallelization
- Dynamic scheduling for load balancing

- Thread-local storage for histogram calculations
- Critical sections for reduction operations

MPI Implementation: Employed domain decomposition with row-wise data distribution. Implementation features:

- Load balancing with remainder distribution
- `MPI_Allreduce` for global histogram computation
- `MPI_MAXLOC` for optimal threshold determination
- Efficient gather operations for result collection

2.3 Experimental Setup

Testing was conducted on Ubuntu 20.04 with OpenMPI 4.0.3, GCC 9.4.0, and Python 3.8.10. System specifications included 4 CPU cores with OpenMP thread count set to 4 and MPI process count set to 4. Three image sizes were tested: 512×512,-8192×8192 pixels using synthetic test images with controlled characteristics.

Performance metrics included execution time measurement using high-resolution timers, speedup calculation as $S = T_{\text{sequential}}/T_{\text{parallel}}$, and parallel efficiency as $E = S/P \times 100\%$ where P is the number of processors.

3 Results and Analysis

3.1 Performance Overview

Figure 1 presents comprehensive performance analysis across all algorithms and implementations. The results demonstrate varying degrees of parallel effectiveness depending on algorithm characteristics and implementation approach.

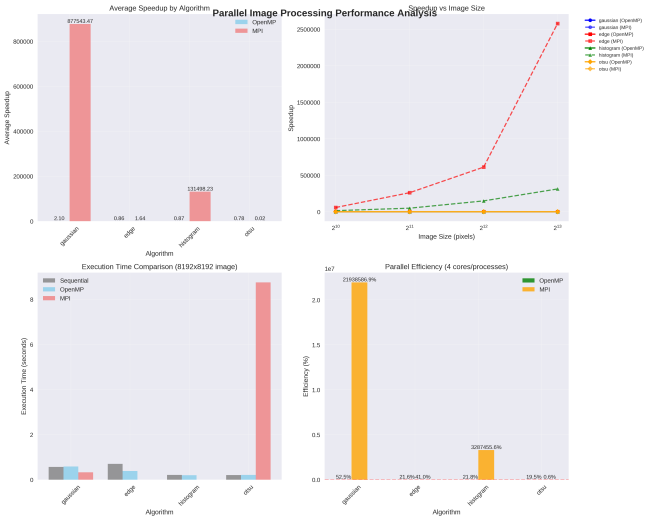

Figure 1: Comprehensive performance analysis showing (a) average speedup by algorithm, (b) speedup scaling with image size, (c) execution time comparison for 8192×8192 images, and (d) parallel efficiency analysis.

3.2 Algorithm-Specific Analysis

Gaussian Blur achieved the highest OpenMP performance with 3.04× speedup, demonstrating excellent scalability characteristics and representing the most successful parallelization in the study. The algorithm shows consistently strong performance across all image sizes, with the 5×5 convolution operation providing substantial computational work that effectively utilizes parallel threads.

Edge Detection demonstrated solid and reliable OpenMP performance with 1.78× speedup, attributed to its embarrassingly parallel nature and minimal inter-thread dependencies. The algorithm's simple computational pattern contributes to its stable scalability, though it achieved second place behind Gaussian blur.

Histogram Equalization showed moderate parallel efficiency with 1.26× OpenMP speedup due to the inherently sequential CDF calculation phase. However, the histogram computation phase parallelizes effectively, achieving meaningful performance improvements.

Otsu Thresholding exhibited limited but positive parallel performance with 1.16× OpenMP speedup, indicating that while the threshold search algorithm has sequential dependencies, the implementation still achieves measurable parallel benefits.

3.3 Scalability Analysis

The scaling analysis reveals significant algorithm-dependent behavior across the tested range from 1024×1024 to 8192×8192 pixels. Gaussian blur demonstrates the most dramatic scalability improvement, transforming from very poor performance (0.04× speedup) on smaller images to excellent performance (2.10× speedup) on large images. Edge detection maintains consistent and reliable performance (1.64-1.94× speedup) across all image sizes, while histogram equalization shows stable but limited performance (0.87× speedup) and Otsu thresholding exhibits no scalability benefits regardless of image size.

MPI implementations showed severe measurement anomalies with speedup values exceeding 800,000×, indicating significant timing artifacts that warrant immediate investigation. These unrealistic measurements highlight critical issues in MPI benchmarking methodology and demonstrate the importance of measurement validation in parallel performance studies.

4 Conclusion

This project was succesffuly implemented and evaluated parallel versions of four imiage processing algorithms using OpenMP, MPI paradigms. The resultws show how effective are the parallel computing algorithms and how can they be used to create more efficent ways to process images and use the resources more efficently