Parallel Image Convolution: Performance Analysis of Edge Detection and Filtering

Muhammad Zayan (2023554)
Muhammad Ahmad Amjad (2023361)
Abdullah Ejaz Janjua (2023038)

Department of Computer Science
CS361L: High-Performance Computing

December 8, 2025

Abstract

This report benchmarks a parallelized Sobel Edge Detection filter utilizing the OpenMP API. We analyzed execution latency, speedup, and parallel efficiency across 1, 2, 4, 8, and 16 threads. By leveraging shared-memory parallelism strategies—specifically loop collapsing and static scheduling the system achieved significant performance gains. The results demonstrate effective scalability on multi-core architectures, with optimal efficiency observed at 8 threads.

1 Introduction

Image convolution is a computationally intensive operation fundamental to computer vision tasks such as edge detection, blurring, and sharpening. Standard serial implementations often suffer from high latency as image resolution increases, creating a bottleneck in real-time processing pipelines.

This project addresses this issue by implementing a multithreaded Sobel operator using OpenMP. The primary objective is to evaluate the speedup and efficiency of parallel execution strategies on a shared memory system, focusing on the trade offs between thread management overhead and raw computational throughput.

2 Methodology

2.1 Algorithm

The Sobel operator calculates the gradient of image intensity using two 3×3 convolution kernels, G_x and G_y. These kernels detect horizontal and vertical edge responses, respectively.

The magnitude of the gradient (G) is computed for every pixel to produce the final edge map using Equation (1):

$$G = \sqrt{G_x^2 + G_y^2} \tag{1}$$

2.2 Parallel Implementation

To optimize performance, we focused on parallelizing the nested loops that iterate over the image dimensions:

- **Loop Collapse:** We utilized the directive `#pragma omp parallel for collapse(2)`. This flattens the nested loop structure (height × width) into a single iteration space, significantly improving load balancing.
- **Scheduling:** We employed `schedule(static)`. Since the convolution workload per pixel is constant (9 multiplications, 8 additions), static scheduling minimizes the runtime overhead associated with dynamic work distribution.
- **Memory Handling:** The `stb_image` library facilitated efficient direct access to raw pixel data, ensuring that the benchmark measured computational performance rather than I/O latency.

3 Performance Evaluation

The benchmarking process involved executing the Sobel filter on a high-resolution test image. We measured the Wall-Clock Time using `omp_get_wtime()`. The performance metrics are defined as:

- **Speedup (S_p):** $T_{serial}/T_{parallel}$
- **Efficiency (E_p):** S_p/p (where p is the thread count)

4 Results

The table below presents the performance data obtained from the benchmark execution.

Table 1: Performance Benchmarking Results (Sobel Filter)

Threads	**Time (s)**	**Speedup (S_p)**	**Efficiency (E_p)**
1	0.013781	1.00x	1.00
2	0.005165	2.67x	1.33
4	0.003161	4.36x	1.09
8	0.002018	6.83x	0.85
16	0.001851	7.45x	0.47

Visual Validation

The figures below demonstrate the visual output of the parallelized Sobel filter applied to the test image.

(a) Original Input Image

(b) Sobel Edge Detection Output

Figure 1: Visual comparison of the input image and the generated edge map.

5 Result Analysis

The quantitative data indicates a clear correlation between thread count and performance improvement:

1. **Scalability:** The system achieves near-linear speedup up to 4 threads (4.36x). As the thread count increases to 8, the speedup continues to grow substantially to 6.83x, confirming that the `collapse(2)` clause effectively utilized the available cores.

2. **Efficiency Drop-off:** Parallel efficiency remains high (> 0.90) for lower thread counts. However, at 16 threads, efficiency drops to 0.47. This is an expected outcome due to Amdahl's Law and the saturation of memory bandwidth, where the overhead of managing threads begins to outweigh the benefits of additional computation power.

3. **Optimal Configuration:** The 8-thread configuration offers the optimal balance between performance (0.002018s) and resource utilization (85% efficiency) for this specific workload.

6 Conclusion

The OpenMP implementation of the Sobel filter successfully demonstrated the benefits of shared-memory parallelism. By utilizing the collapse directive and optimizing memory access patterns, we achieved substantial speedups compared to the serial baseline. The results validate the effectiveness of parallel processing for computationally intensive image processing tasks, while also highlighting the diminishing returns observed at higher thread counts due to hardware constraints.