# Benchmark Report for Convolution Parallelization Using MPI

Luc Pereira Cardoso, Parsa Vares, Tom Walter

December 2, 2024

## 1 Introduction

This report presents the results of benchmarking a convolution operation. We used the provided sequential implementation and developed a parallelized version. The goal was to achieve significant speed-up and maintain the correctness of the output.

## 2 Parallelization Approach

We divided the input data into chunks distributed across available processes using MPI. Each process performed the convolution operation on its assigned chunk. The results from all processes were gathered at the root process for final assembly.

## 3 Benchmark Results

The benchmark results were measured in real-time and CPU time. Each run was executed 30 times for both the sequential and parallel versions. The average and standard deviation values for each metric were recorded.

### 3.1 Sequential Execution

The sequential execution results are as follows:

- Real Time: Average = 1.620217 seconds, Standard Deviation = 0.007532 seconds.

- CPU Time: Average = 1.611895 seconds, Standard Deviation = 0.007905 seconds.

### 3.2 Parallel Execution

The parallel execution results with varying numbers of processes are shown in Table 1. (RT - Real Time, Chunk - Chunk Size, CPU - CPU Time). The image format was 28 rows so we chose process amounts that were divisible by 28. In the following table you see 28 Processes (1 Chunk per process(Cpp)), 14 (2 Cpp) and so on.

Table 1: Benchmark Results for Parallel Execution

| **Chunk** | **RT Avg (s)** | **RT Std (s)** | **CPU Avg (s)** | **CPU Std (s)** |
|---|---|---|---|---|
| 1 | 0.0609 | 0.0003 | 0.0591 | 0.0005 |
| 2 | 0.1196 | 0.0153 | 0.1176 | 0.0153 |
| 4 | 0.2325 | 0.0159 | 0.2301 | 0.0157 |
| 7 | 0.4043 | 0.0154 | 0.3971 | 0.0157 |
| 14 | 0.7929 | 0.0146 | 0.7885 | 0.0145 |

## 4 Speed-Up Calculation

The speed-up gain was calculated as the ratio of the sequential real-time execution average to the parallel real-time execution average for each configuration. The results demonstrate significant performance improvement:

$$\text{Speed-Up} = \frac{\text{Real Time (Sequential)}}{\text{Real Time (Parallel)}}$$

- **1 chunk:** Speed-Up $= \frac{1.620217}{0.060880} \approx 26.61$
- **2 chunks:** Speed-Up $= \frac{1.620217}{0.119625} \approx 13.55$
- **4 chunks:** Speed-Up $= \frac{1.620217}{0.232486} \approx 6.97$
- **7 chunks:** Speed-Up $= \frac{1.620217}{0.404292} \approx 4.01$
- **14 chunks:** Speed-Up $= \frac{1.620217}{0.792913} \approx 2.04$

Similarly, CPU time speed-ups were calculated:

$$\text{Speed-Up} = \frac{\text{CPU Time (Sequential)}}{\text{CPU Time (Parallel)}}$$

- **1 chunk:** Speed-Up $= \frac{1.611895}{0.059065} \approx 27.29$
- **2 chunks:** Speed-Up $= \frac{1.611895}{0.117615} \approx 13.70$
- **4 chunks:** Speed-Up $= \frac{1.611895}{0.230122} \approx 7.00$
- **7 chunks:** Speed-Up $= \frac{1.611895}{0.397057} \approx 4.06$
- **14 chunks:** Speed-Up $= \frac{1.611895}{0.788539} \approx 2.04$

These results show the parallel implementation's scalability, improving speed-up as more processes are used. Given the nature of this task having more kernels working at the same time will result in shorter execution time.

# 5 Output Quality Comparison

To ensure that the parallel algorithm produces results of the same quality as the sequential implementation, we compared the denoised images output by both approaches. The visual inspection revealed that both methods effectively filtered out the noise.

Figures 1 and 2 show the denoised images from the sequential and parallel algorithms, respectively.

# 6 Conclusion

The parallelization of the convolution operation using MPI showed very good scalability and correctness. By distributing the workload among multiple processes, we achieved a significant reduction in execution time.

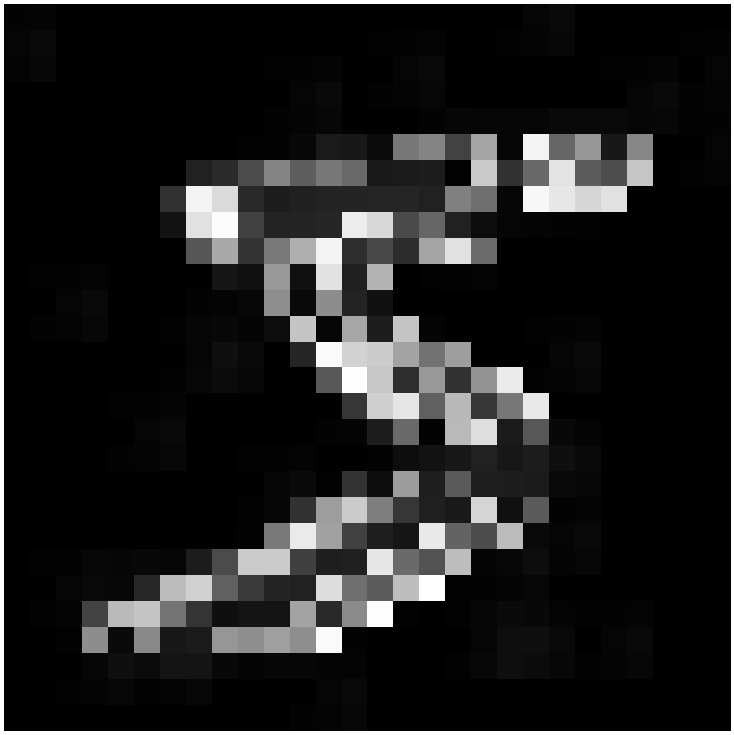

Figure 1: Denoised image from the sequential algorithm.

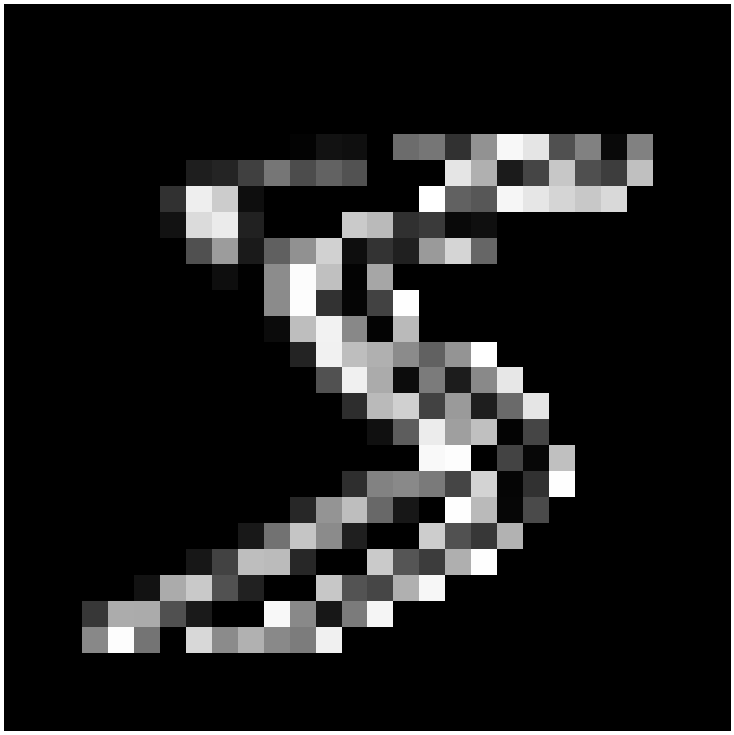

Figure 2: Denoised image from the parallel algorithm.