# High Performance Computing Assignment: CUDA vs MPI vs OpenMP Performance Analysis

Kisalay Ghosh

March 26th 2025

## Introduction

This report presents a comparative analysis of three parallel computing approaches—OpenMP, MPI, and CUDA—used to convert a color image to grayscale. The assignment required implementing a CUDA program and comparing its performance to existing MPI and OpenMP implementations.

## CUDA Implementation Overview

The CUDA implementation uses GPU parallelism to assign each pixel to an individual thread, achieving high parallel throughput. The image is loaded into host memory, transferred to device memory, processed on the GPU, and written back to host memory.

### CUDA Kernel

Listing 1: CUDA Kernel to convert RGB to Grayscale

```
__global__ void grayscale_kernel(unsigned char *input, unsigned char *output,
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < width && y < height) {
        int idx = y * width + x;
        int i = idx * 3; // 3 channels: RGB
        unsigned char r = input[i];
        unsigned char g = input[i + 1];
        unsigned char b = input[i + 2];
        output[idx] = (unsigned char)(0.299f * r + 0.587f * g + 0.114f * b);
    }
}
```

# How to Run the Assignment

1. Place the input image in the working directory as `input.jpg`.
2. Compile the CUDA program:

```
nvcc cuda_grayscale.cu -o cuda_grayscale
```

3. Run the executable:

```
./cuda_grayscale input.jpg output.jpg
```

4. The output image will be saved as `output.jpg`.

Table 1: Execution Times (in seconds) for OpenMP, MPI, and CUDA. This data is taken from the previous assignemnts

| Threads / Procs | OpenMP | MPI | CUDA |
|---|---|---|---|
| 1 | 0.003044 | 0.00125 | 0.000368 |
| 2 | 0.000446 | 0.00140 | 0.000368 |
| 4 | 0.000258 | 0.00070 | 0.000368 |
| 8 | 0.000230 | 0.00090 | 0.000368 |

# Graphical Analysis

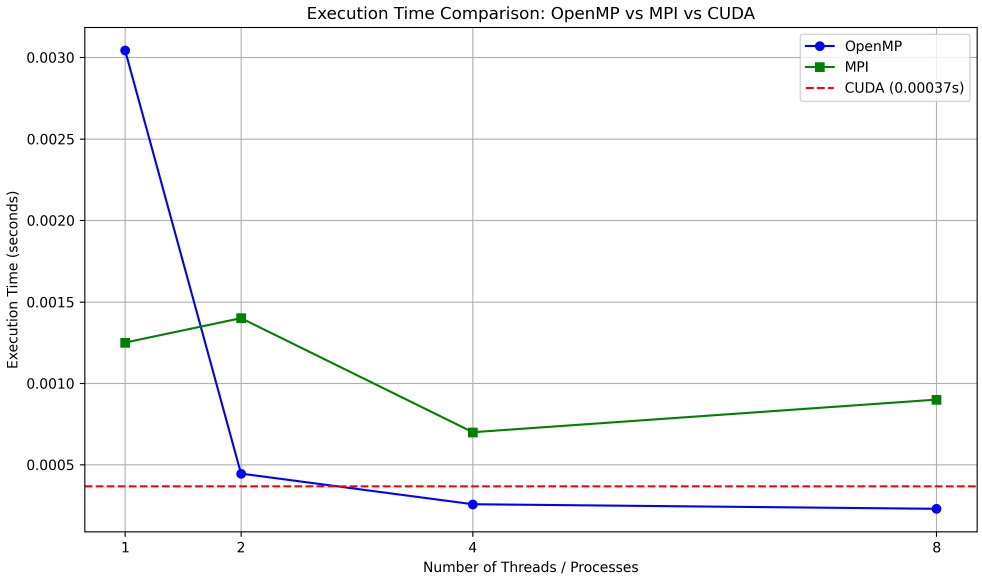


Figure 1: Execution Time Comparison: OpenMP vs MPI vs CUDA

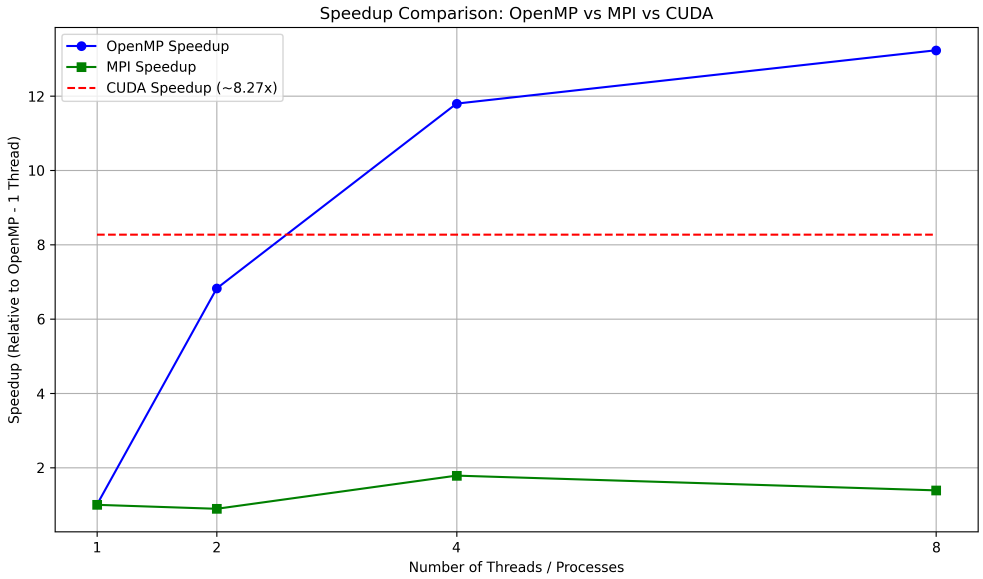


Figure 2: Speedup Comparison: OpenMP vs MPI vs CUDA

The first figure shows that CUDA outperforms MPI and OpenMP (except at 8 OpenMP threads). CUDA execution time remains constant due to massive parallelism. The second figure reveals that OpenMP scales better than MPI, and CUDA achieves ˜8.27x speedup over single-thread OpenMP.


Figure 3: CUDA Output Terminal Showing Execution Time and Save Confirmation

The first figure shows that CUDA outperforms MPI and OpenMP (except at 8 OpenMP threads). CUDA execution time remains constant due to massive parallelism. The second figure reveals that OpenMP scales better than MPI, and CUDA achieves ˜8.27x speedup over single-thread OpenMP.

# Difficulties Faced

- Initial image output was distorted due to incorrect flipping and stride.
- Debugging image orientation was time-consuming and involved checking flip logic in `stb_image`.
- Compiling with CUDA under WSL required ensuring correct driver and device support.
- Comparing MPI and CUDA required careful data synchronization and measuring GPU timings accurately.

# Conclusion

CUDA provides significant performance gains due to GPU-level parallelism. OpenMP scales well with threads but is limited by CPU cores. MPI shows performance degradation with too many processes. For image processing tasks involving large-scale pixel operations, CUDA is highly efficient.