INF3201 - Parallel Programming

Assignment 2 - Parallel Raster Image Filtering using MPI and OpenMP

September 15, 2025 - October 06, 2025

1 Raster Image Filtering

Image filtering is a common digital image processing technique used to enhance or reveal particular features of an image. Figure 1 depicts two types of convolution based filter [3] (Emboss and Sobel) applied to an image.

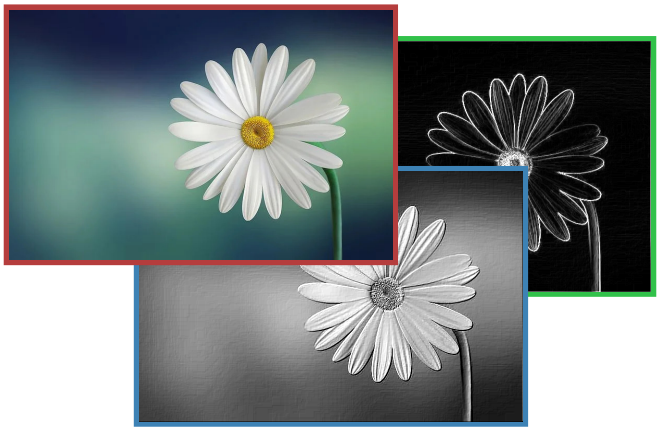

Figure 1: Image samples. Red is the original image. Blue one is the original image with an Emboss filter applied to it. Green image is the original image with a Sobel filter applied to it.

Here is an example of how to apply the Sobel filter on an image. First, the RGB (Red, Green and Blue channels) image are converted to grayscale. This can be achieved by using the luminosity method [6] by assigning the following *grayscale* value to each channel $grayscale = 0.3R + 0.59G + 0.11B$. Next, two 3 by 3 matrices (or kernels) should be convoluted with the grayscale version of the image A:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * A \qquad G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * A$$

The final value of each pixel is defined by the matrix $G = \sqrt{{G_x}^2 + {G_y}^2}$. Figure 2 details the steps to apply the Sobel filter on a single pixel of a grayscale image.

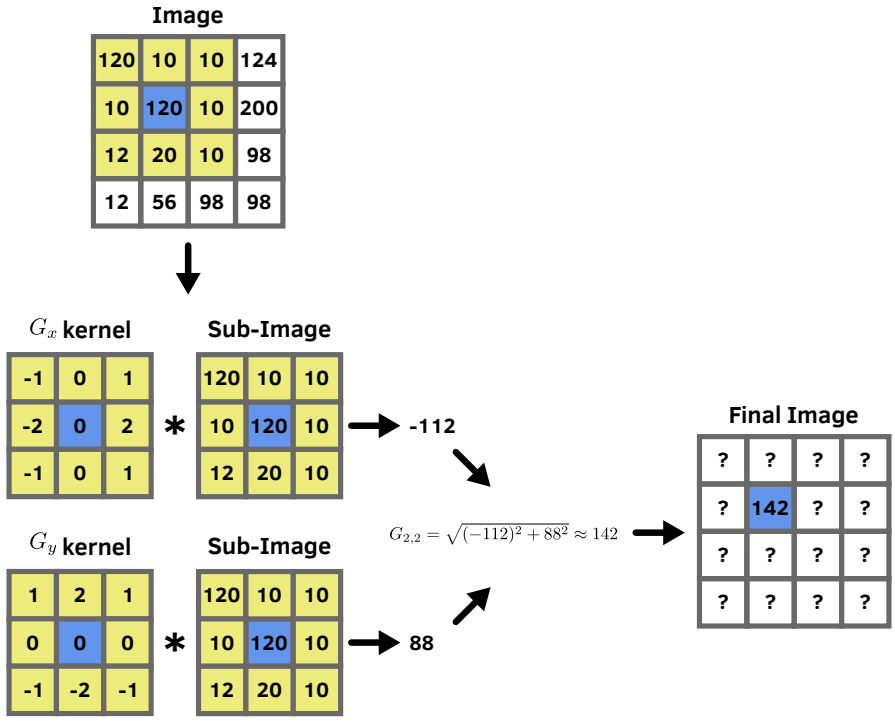

Figure 2: Sobel filter applied to a single pixel of a grayscale image. Note that $*$ denotes the 2-dimension signal processing convolution operator [7].

2 Image File Formats

Many raster image file formats are available [4]. In this assignment you will use the BitMaP image files (or BMP). It is one of the simplest raster image file formats. For the sake of simplicity, a simple BMP library (that comprise **bmp.h** and **bmp.c**) is given. This library is a wrapper around **libbmp**[1] and has the following functions:

```
void GetSize(const char* filepath,int* width, int *height);
void LoadRegion(const char* filepath,const
                int x,const int y,const int width,const int height,RGB* region);
void WriteRegion(const char* filepath,
                 const int x,const int y,const int width,const int height,RGB* region);
void CreateBMP(const char* filepath,const int width,const int height);
```

See the source files for a more in-depth understanding. When using your own images, be careful to use a BMP format that is compliant with the **libbmp**[1] library.

Note that **libbmp** do not handle *color space information*. If you use your own images, be careful to remove these information. To remove these information you can use Gimp and check the box "Do not add color space informations" during export.

One solution to have larger images is to combine several ones. If you have *ImageMagick* installed on your machine, you can combine 2 images *a.bmp* and *b.bmp* with the following command:

```
> convert a.bmp b.bmp -append out.bmp
```

3 Task

In the first place, you have to implement a sequential image filtering program that can apply an Emboss and a Sobel filter to an image.

Next, you have to implement a parallel image filtering program written in C/C++, using MPI and OpenMP. The goal is to parallelize the filtering of **one image** with MPI and parallelize the computation on each MPI process using OpenMP. Two filters must be implemented: Sobel and Emboss. You are not allowed to use any additional libraries (aside from MPI, OpenMP and the BMP library mentioned in the last section).

In the parallel version, you must divide the original image into tiles that are distributed to the processes (at least one tile per process). Each process applies the given filter to its tile(s). Your algorithm must account for potential missing pixels.

Each version of your code should be tagged, saved and accessible for evaluation. Once this version is implemented, you should propose (at least one) optimization to your code. Each optimization should be backed up by evaluations, against the initial parallel version.

To evaluate the performance of your implementations, you need to run a set of measurements. We will focus on time measurements. The sequential version will be a baseline for comparison against your parallel version(s), running on multiple nodes. Your solution should be able to scale when size and the number of processes vary.

4 Requirements

4.1 General

- A high vigilance is given to plagiarism. Always reference when you are using resources. It is not allowed to use other's code or code generators.
- Your solution can not use any shortcuts that reduce the functionality of the program nor the difficulty of the assignment. For example, do not use any available mechanism on the cluster that would allow you to not use MPI and still provide a solution (e.g a shared file system). If you have a doubt, ask the TA during colloquiums.
- Each chosen communication pattern should be backed up and explained in the report. Attention is put on the quality of the solution, with regards to the problem and input given in this document.
- Write code using C/C++, MPI and OpenMP.
- Make sure the code is well-commented, in English.
- Make sure to provide a README file that explains how to run your sequential and parallel codes.
- Ensure that your MPI processes run on the machines that you carefully selected using a *hostfile*

4.2 Time performance analysis

Evaluate the time performance of your parallel version. For that, you should study two dimensions: the number of processes, the size of input. Have a graphical representation of your two studies (number of processes and size of input on the x axis and time to solution on the y axis). These two separate graphs help you answer the following questions: how well does your solution(s) scale, according to the number of processes (for the highest studied image size)? How well does your solution(s) scale according to the size of the image (for a fixed number of processes)?

For OpenMP, try to quantify the impact of the number of threads for a fixed, and carefuly chosen problem size and number of processes. For that, have a graph or a table that shows how well does your solution(s) scale, according to the number of threads.

4.3 Report

The report should be between 6 and 10 pages. Remember to explain everything in detail and answer questions from the assignment. Make your figures clear and understandable. Include references, captions and axes descriptions. Take into consideration the hardware used (e.g heterogeneity). The report should have the following sections:

- *Introduction*
  - Explain your understanding of the assignment.
- *Sequential solution*
  - Explain your sequential solution and how to run it.
- *Parallel design solution*
  - Explain how you parallelized and optimized your program with MPI and OpenMP.
  - Describe how you parallelized the code, how the workload distribution is being done. Explain your optimized version(s). Explain how to run each version.
- *Time Performance analysis*
  - Detail the hardware you used for testing
  - Compare the serial and parallel version of your program (speedup and efficiency).
  - How shared and distributed memory helped in improving the performance in your work?
- *Discussion*
  - Discuss positive and negative points of your solutions. Compare the theoretical maximum speedup with your results with different sizes while using different number of processes starting from 2 processes to the maximum available number of processes in the cluster. This is also the section to discuss the proposed optimizations done to your code.
- *Conclusion*
  - Summarize and conclude your work.

5 Archive

The *code/* directory contains:

- *serial/* - Source code of the serial program
- *parallel/* - Source code of the parallel program

The *serial/* and *parallel/* folders contains:

- *lib/* - Libraries used for the assignement
- *lib/libbmp/* - Library to load BMP files (you do not need to understand the source code in this folder)
- *lib/bmp.h* - Interface of the BMP library
- *lib/bmp.c* - Implementation of the BMP library
- *Makefile* - Receipts to build and run your program

6 uvcluster

The Computer Science department has a cluster of nodes that should be used to run your solution. You need to copy your files only to the frontend (ificluster.ifi.uit.no) to access them from all other nodes. In order to login to the cluster, use (in linux):

```
ssh <your UiT ID>@ificluster.ifi.uit.no
```

Make sure that you are familiar with the welcome message from the cluster, helping you to use the cluster correctly.

7 Hand-in

You will work by group of 2 for this assignment. GitHub classroom is used as a hand-in platform for the course. You can and probably should commit and push as often as possible. **Remember to push all your changes before 06 October 2025 23:59:59**. Any changes pushed to your repository after that deadline will not be evaluated.

References

[1] https://github.com/marc-q/libbmp

[2] https://en.wikipedia.org/wiki/BMP_file_format

[3] https://pro.arcgis.com/en/pro-app/2.8/help/analysis/raster-functions/convolution-function.htm

[4] https://en.wikipedia.org/wiki/Image_file_format#Raster_formats

[5] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[6] https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

[7] https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution