INF560 · Image Filtering

Delacourt Remi · Amponsem Kwabena

March 15, 2021

Contents

1 Introduction

In this project, parallel computation techniques were used to improve the performance of an image filtering program. In the sequential version, a GIF image is accepted as input and then an output, after filtering, is generated.

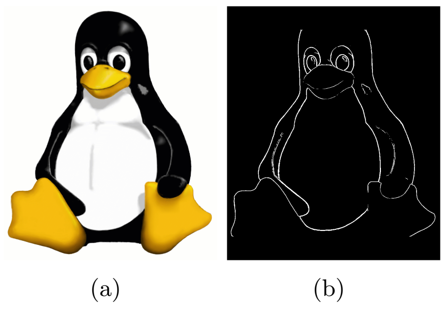

Figure 1: (a) Original GIF image (b) Processed GIF image

To parallelize the sequential version, several parallelization models were studied and after vigorous investigation, one of the models were chosen.

2 Overview

Data parallelism is the type of parallelism employed in this project as it is the main focus of this module, INF560.

The pixels of an animated GIF image are distributed in a fashion over all the processes. Initially, splitting animated GIF images into individual images was considered.

At the first glance, that model seemed like the best option but after several iterations of critical thinking, it was noticed that the model only works best when there is an equal of amount of images as there are processes. Programming for such ideal cases had to be avoided at all cost since the application would be used in a real world environment where ideals are hard to achieve.

The report is structured as follows:

- An in-depth explanation of the implementation of the parallel algorithm shown in Figure 2 (considering MPI, OpenMP, and CUDA.)
- Performance evaluation juxtaposing the parallel version against the sequential version.
- A conclusion expressing how the objectives of the projects were achieved.

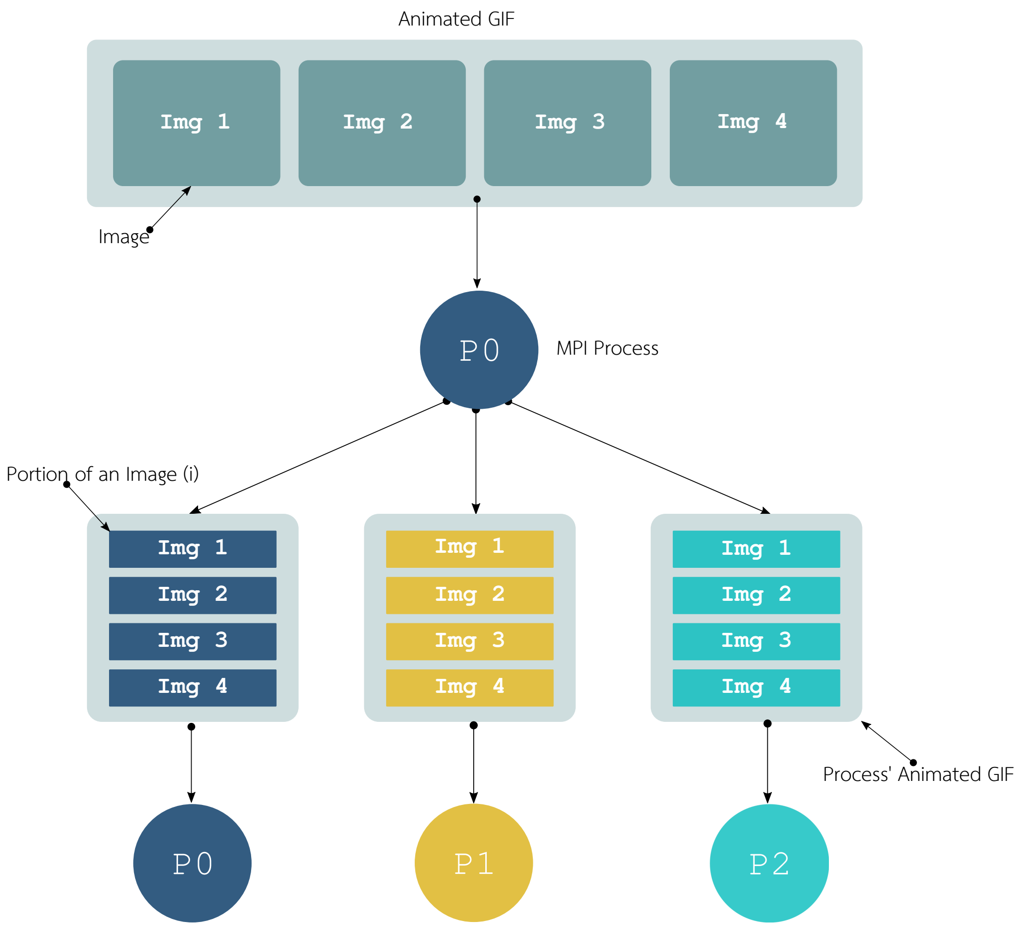

Figure 2: High-level abstraction of the parallel algorithm.

3 Implementation

In this section, an in-depth explanation of the parallel algorithms employed in the project: how animated images were split up; how the images were recombined; how processing occurs on each process, is provided.

In our model, each image in the animated GIF is split according to the number of processes. This means that considering Figure 2, since there three (3) processes, each *Img* would be divided into three portions along the height of the image, color coded as *dark blue*, *yellow*, and *light blue*.

In that sense, each process would have a new animated image consisting of smaller chunks of all the individual images in the animated image.

3.1 MPI

The main focus of parallelism with MPI in this project is to parallelize computation and not I/O operations, and so, one process, **Process 0**, loads each animated GIF image and broadcasts its pixels to all the other processes.

Once a process receives the loaded pixels and metadata i.e., the number of individual images within the animated GIF, it then calculates the portion it has to work on. With this approach, each process calculates its own portion to work on.

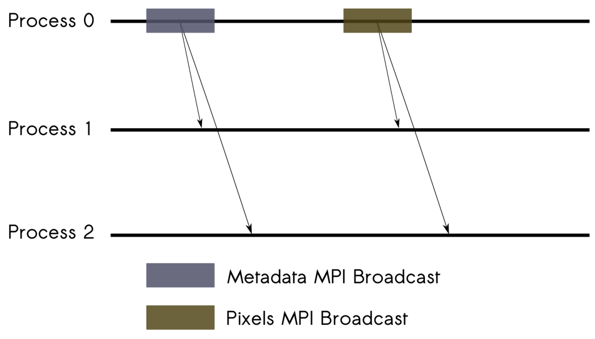

Figure 3: The master process sending broadcasting data to other processes.

3.2 OpenMP

3.3 CUDA

4 Evaluation

5 Conclusion

In this project, several approaches were initially studied and then one i.e., the algorithm illustrated in Figure 2, was decided upon and implemented. And the results showed that the performance increased with increasing number of processes and threads.

This is the link to the GitHub repository of the project: