Real-Time Operating Systems

INFO-F-404

Parallel computing project : Image processing

26 November 2020

HAUWAERT Maxime : 000461714

Université Libre de Bruxelles (ULB)

Contents

1 Introduction

In this project it was asked to create an efficient platform that will allow people to easily blur other people's faces. This platform will split the work between multiple processors in order to reduce the execution time. It will use MPI calls to communicate between the processes.

2 Concepts

In order to fully understand this report a few concepts have to be defined first :

- **MPI** : Message Passing Interface. it is a message-passing standard used by different computers or in a multiprocessor computer to communicate between each other. It simplifies the development of multiprocessor programs. The most common used strategy is a main process that splits the work between the different processes and then gather all the partial results.
- **Average Blur** : It is a blurring technique where each pixel to blur is equal to the average of all pixels in its neighbourhood of a specified size. It is mainly used to hide details so that the viewers cannot distinguish precisely certain parts of the images. In the context of this project, the parts to blur will be the people's faces.

3 Description

The program takes as arguments the path to the raw grayscale image, the path where to save the resulting image, the path to the masks file containing all the masks on which to apply the blurring effect, the size of the neighbourhood to use for the blurring method and eventually the dimensions of the image (it is by default 1280 by 720). The program will use the open source MPI library to split the work and to communicate between the different processes.

4 Implementation

4.1 Information

The source files are written in C and are located in the `source` folder.

The code can be launched by executing this command : "mpirun -n 4 blur" followed by the options.

As C is not an object oriented programming language, in order to have a more readable code it has been decided to simulated objects with structures and methods with functions that takes the pointer to the structure. The "h" files contains the declarations and the "c" files contains the definitions.

4.2 Libraries

All the libraries used in this project are standard C libraries. Here is the list of all the libraries used in this project : stdio.h stdlib.h stdbool.h unistd.h. These libraries are used for different purposes such as printing strings, parsing arguments from the command line, defining the bool type, allocating space in the memory and more.

4.3 Work dispatch

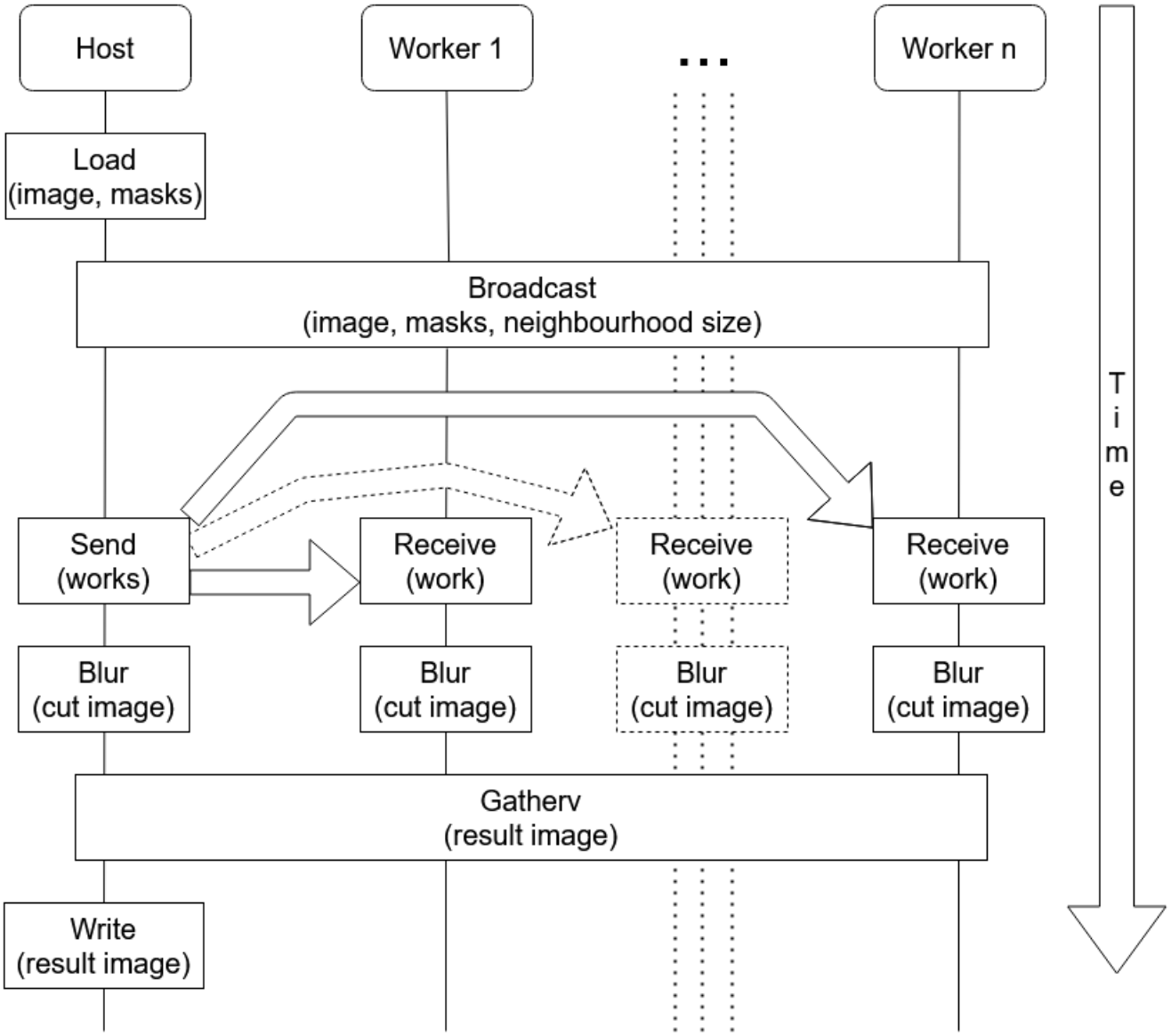
The strategy used with MPI in this project is the following:

- The host analyses all the parameters and loads the image and the masks.
- The host sends all the parameters to the workers via broadcast.
- The workers receive all the parameters via broadcast.
- The host divides the work by giving each process(including himself) which rows they have to work on via multiple send instructions.
- The workers receives their tasks via recv instruction.
- Each process works on their tasks.
- When finished each process sends its result to the host via gatherv.
- The host collects all the different parts of the image, via gatherv, and writes the result in the specified file.

Notes :

- The host is the process with rank 0 and the workers are all the others.
- gatherv is a special version of gather where the processes do not send necessarily the same number of units of data.

Here is an abstract schema representing the strategy described above for an undefined number of workers.



As the schema is abstract it does not show technical aspects of the code. For example to send the image it is needed to first send its width and its height to the processes in order to allocate enough space in the memory and to know the number of bytes they should receive.

4.4 Structures

Here are the structures used in this project with the functions linked to them. The structures have the same name as the file they are defined in.

4.4.1 Image

This struct represents an image.
It has a width, an height and the data.
Here are the main functions :

- `load_raw_image` : it loads the image from the specified raw file.
- `get_blur_pixel` : it calculates the value of the pixel at the specified position of the image by applying the average blurring method.

4.4.2 Mask

This struct represents a masks.
It has a start position and an end position
Here are the main functions :

- `get_all_masks` : it loads all the masks contained in the specified file.
- `is_pixel_in_masks` : it verifies that the specified pixel is in at least one of the masks.

4.4.3 Options

This struct represents the set of options to run the program.
It has the input image path, the output image path, the masks file path, the size of the neighbourhood, the width and the height of the image.
Here is the main function :

- `parse_arguments` : it parse all the options of the command line.

4.5 Blur

It is the main file. It contains the code for the host and the workers. Here are the main functions :

- `blur_host` : it executes the code of the host.
- `blur_worker` : it executes the code of a worker.
- `create_mpi_mask_type` : it creates a new type for MPI to make the receiving and the sending of masks easier.
- `balance_work` : it splits the quantity of work evenly between the different workers, if the rest of division of the quantity of work by the number of workers is not equals to zero, the first workers receive one additional unit of work.

4.6 Blurring borders

In the definition of the average blurring method a problem arises, a part of the neighbourhood of pixels near the borders goes beyond the limits of the image. To solve this problem it has been decided to just take the part of the neighbourhood that is in the image and to discard the other part. So the number of pixels taken in the computation of the average is variable.

4.7 Masks file

In these files each line corresponds to a mask. On each line there are four values, the first two values correspond to the position of the top left corner and the last two values correspond to the position of the bottom right corner of the mask.

4.8 Indices computation

The images are stored in one dimension but the images are normally defined in two dimensions. So in order to get the index in the array of the pixel located at the position (i,j) the following computation is needed : $i * image.width + j$.

4.9 Makefile

The makefile allows the users to easily build the program, clean the useless files (created during the building) and test the different examples. The users just need to update, if needed, the path to the directory where openmpi library is located.

4.10 Miscellaneous

- When the masks file is empty it has been decided to directly stop the program as the output image will not be different than the input image, it would be a waste of resources.
- Variable-lenght array have not been used as they can be dangerous, for example, as the user can give the dimensions of the image, if the dimensions are to big it is not possible to properly handle memory problem with variable-lenght array.

5 Errors

Every errors that might happen have been handled. When an error occurs the program prints a message describing the error and terminates all the processes.

6 Examples

In order to let the users test the program four example files have been added with their corresponding masks files and everything is located in the `example` folder. The makefile contains all the examples for the users to try.

7 Tests

There are not a lot of interesting tests to make as when executing the examples all the possible remaining errors should appear because nothing really changes between different instances of the program. The only worth trying example is when the height of the image cannot be entirely divided by the number of processes, it means that all the processes do not have the same number of rows to work on. As the gatherv instruction is far from being the easiest one to use, it might be the main sources of bugs.

8 Problems encountered

The only real problem encountered was that C is not an object oriented programming language so in order to have a readable code the chosen solution was to simulate objects with structures and methods with functions that takes the pointer to the structure (sort of `this`).

9 Conclusion

This program is a great and efficient tool based on the message passing interface that allows users to blur other people's faces. The program takes advantages of multiprocessor architectures in order to reduce its execution time.

A User Manual

A simple and efficient platform to blur people's faces in raw grayscale images.		
blur <-i input_image> <-o output_image> <-m masks_file> <-n number> [-d number number]		
-i input_image	Specify the path of the input image.	The path can be absolute or relative to the current working directory.
-o output_image	Specify the path of the output image.	The path can be absolute or relative to the current working directory.
-m masks_file	Specify the path of the file containing the masks.	The path can be absolute or relative to the current working directory.
-n number	Specify the time limit of the simulated execution.	Valid values are all the natural numbers above 0.
-d number number	Specify the dimension of the input image.	Valid values are all the natural numbers above 0. The width is followed by the height. By default 1280 720.
< > = mandatory [ ] = optional		

Notes : This program should be launched with the MPI program
"`mpirun -n <number of cores>`" followed by the program described in this user manual
Example : `mpirun -n 4 blur -i example/police1.raw -m example/mask1 -n 10 -o test1.raw -d 1280 720`

Warning MPI must be installed