Efficient Parallel Grayscale Image Processing: Unleashing the Power of Parallel Programming

1st Margaritis Pasxalis
University of Western Macedonia
Electrical and Computer engineering
Kozani, Greece
ece01671@uowm.gr

2nd Nikou Rafail
University of Western Macedonia
Electrical and Computer engineering
Kozani, Greece
ece01695@uowm.gr

3rd Fragos Panagiotis
University of Western Macedonia
Electrical and Computer engineering
Kozani, Greece
ece01753@uowm.gr

***Abstract*—Nowadays, a very well-known technology used in the field of study of Computer Vision is image segmentation. This technique is about divided, making into segments an image for the purpose of making the process faster and better. Also, this technique, is very popular in image recognition and image tracking. More precisely, one stage used by a variety of segmentation algorithms involves the conversion of a colored image to grayscale. The colors of each pixel are represented by the popular values. Each parameter (red, green, and blue) defines the intensity of the color with a value between 0 and 255. For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255), and the other two (green and blue) are set to 0. There are many techniques that allow as to convert the RGB values of an image to grayscale, for example a popular one is by using a weighted average of the apparent brightness of each color component. In this paper, we are going to use "The luminosity method" to grayscale an image in parallel programming and flip it horizontally or vertically. We will also compare our parallel implementation to the serial one and understand the benefits of parallel techniques.**

***Index Terms*—Color segmentation , RGB image, brightness, grayscale,pixel,luminosity**

I. Introduction

The coloring that the human eye is able to understand is represented by a combo of three components called RGB values. More precisely, each digital image is represented using values of three primary colors red, green, and blue making the RGB color environment. The other colors occur by a mixture of these primary colors. A color image is converted to a monochrome image by reducing a three-dimensional representation to a unidimensional one and mapping brightness values to shades of gray. The most common approach converts luminosity using a weighted sum in the following format:

$$Y_i = a_1 r_i + a_2 g_i + a_3 b_i,$$

where (ri, gi, bi) stand for RGB components and (1, 2, 3) are the weights of each component.

II. Related Work

A. Previous work

Outwardly vital picture highlights regularly vanish when color pictures are changed over to grayscale. The calculation presented decreases such misfortunes by endeavoring to protect the notable highlights of the color picture. The Color2Gray is a technique often used in parallel programming with GPU process units. The Color2Gray calculation may be a 3-step handle: 1) change over RGB inputs to a perceptually uniform CIE L*a*b* color space, 2) utilize chrominance and luminance contrasts to form grayscale target contrasts between adjacent picture pixels, and 3) illuminate an optimization issue outlined to specifically tweak the grayscale representation as a work of the chroma variety of the source picture. The Color2Gray comes about offer watchers notable data lost from past grayscale picture creation strategies.

B. The luminosity method

This method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. Through many repetitions of carefully designed experiments, psychologists have figured out how different we perceive the luminance or red, green, and blue to be. They have provided us a different set of weights for our channel averaging to get total luminance. The formula for luminosity is:

$$Z = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$

According to this equation, Red has contributed 21 % percentage, Green has contributed 72 % which is greater in all three colors and Blue has contributed 7% .

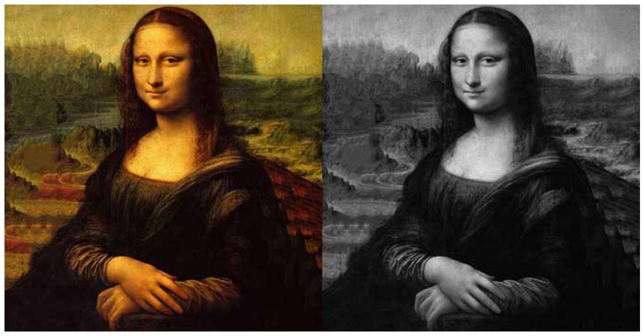

Fig. 1. Grayscaling Mona Lisa.

C. Parallel Programming

Parallel programming is the method of part an issue into littler assignments that can be executed at the same time –

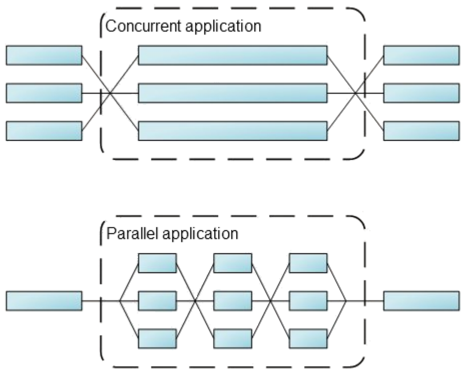

Fig. 2. Parallel Programming example.

in parallel – utilizing numerous computing assets. In other words, parallel programming permits software engineers to run large-scale ventures that require speed and exactness. Moreover, multithreaded programming could be a subset of parallel programming in which more than one set of sequential instructions (“thread”) executes concurrently. Multithreading could be a concept that can exist either on a single center, or numerous forms. On the off chance that the strings are run on a single processor, the processor quickly switches between the strings. It’s vital to point out that on a single center, quickly exchanging forms isn’t a genuine representation of multithreaded programming, but more a case of the CPU prioritizing the execution of these forms. When the strings run on different processors, they’re executed at the same time.

III. Code Implementation

A. Methodology

Step 1: The target colour image is read,
Step 2: Each rank will take a portion assigned to it by the following formula

- start-index = rank * chunksize
- end-index = (rank + 1) * chunksize

where: chunksize = totalpixel/size,
size = number of total ranks,
Step 3: Each rank stores the values of the pixel it reads from the original bmp image in a local array after applying the following formula to grayscale the value of the pixel.

$$\text{new-colour} = 0.2126 \cdot \text{red} + 0.7152 \cdot \text{green} + 0.0722 \cdot \text{blue}$$

Step 4: The root rank gathers all the other arrays into one big 1-D array
Step 5: For each pixel in the 1-D array, depending on whether we want to flip the image or not, we read the value and set the value of the corresponding pixel of the bmp image to that.
Step 6: Stop the time.
Step 7: Save the bmp image
Step 8: Print the time.

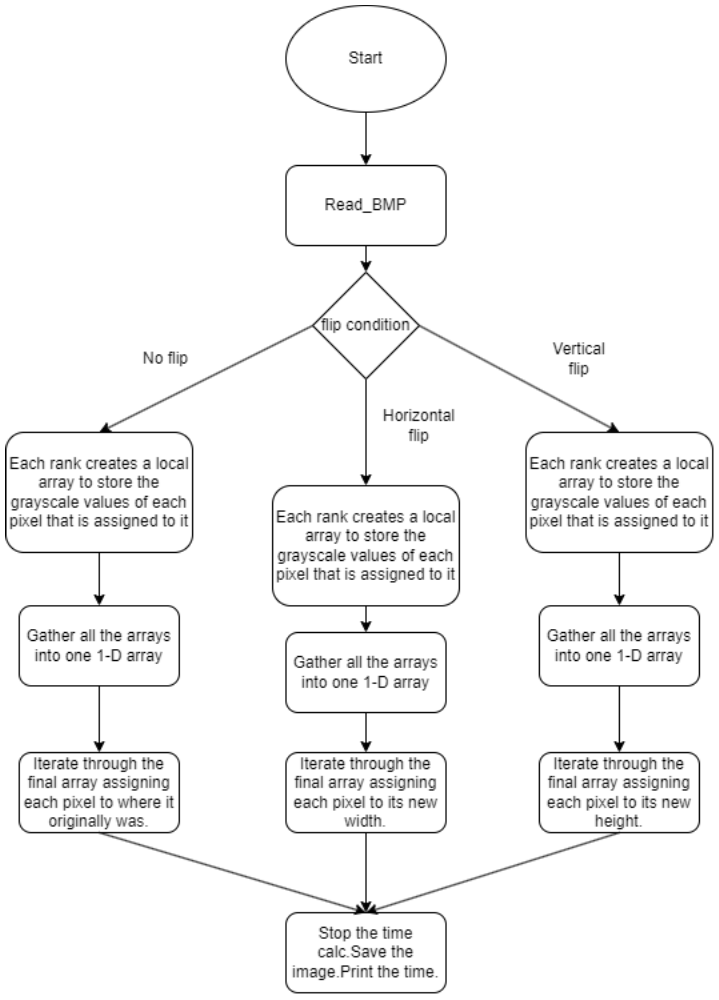
Start
Read_BMP
flip condition
No flip
Horizontal flip
Vertical flip
Each rank creates a local array to store the grayscale values of each pixel that is assigned to it
Each rank creates a local array to store the grayscale values of each pixel that is assigned to it
Each rank creates a local array to store the grayscale values of each pixel that is assigned to it
Gather all the arrays into one 1-D array
Gather all the arrays into one 1-D array
Gather all the arrays into one 1-D array
Iterate through the final array assigning each pixel to where it originally was.
Iterate through the final array assigning each pixel to its new width.
Iterate through the final array assigning each pixel to its new height.
Stop the time calc.Save the image.Print the time.

Fig. 3. Implementation FlowDiagramm.

B. Implementation - Results

We can see from these results that the code used for each implementation does not provide the most optimal results. In the Mpi implementation because the root rank has to do the final work of changing each pixel value while reading the array with the modified pixel values we get a bottleneck. In addition with a lot of processes the cost to communicate and send the data to the root node ends up taking more than the actual processing of the bmp image does. In the omp implementation we can see that the results we get are better but after a certain point it provides little difference adding more threads. In the hybrid implementation the results are around the same where adding more processes or threads after a certain point provides little difference and sometimes even worse results.

Conclusion

In our paper, “Efficient parallel grayscale Image processing: unleashing the power of parallel programming”, we have discussed how parallel programming techniques can be applied to grayscale picture processing using luminosity method in detail. We will summarise the main findings and conclu-

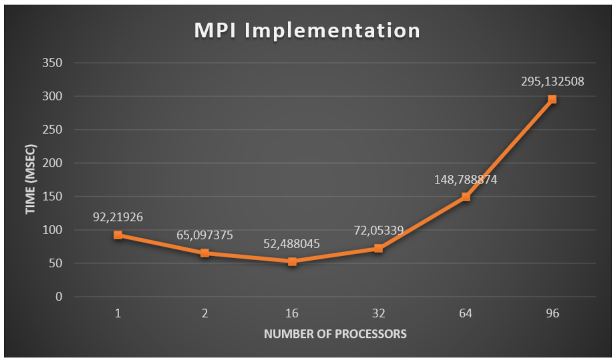

Fig. 4. MPI implementation

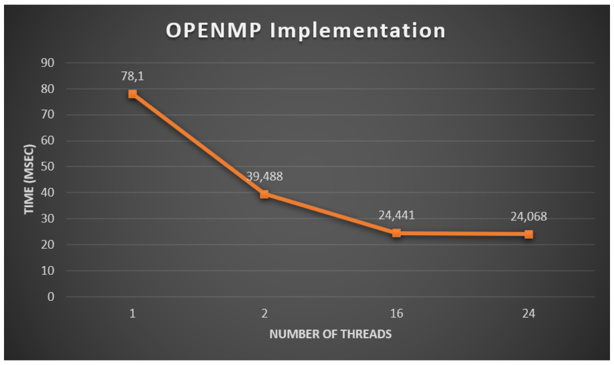

Fig. 5. OPENMP Implementation

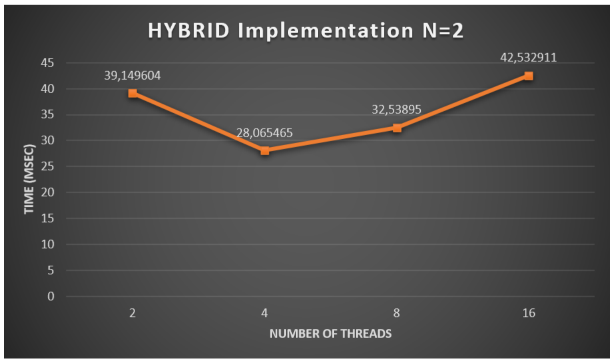

Fig. 6. Hybrid Implementation N=2

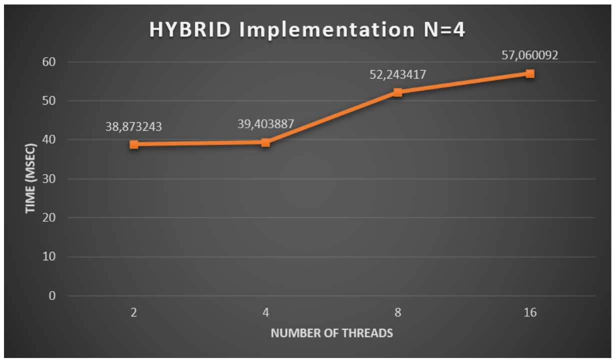

Fig. 7. Hybrid Implementation N=4

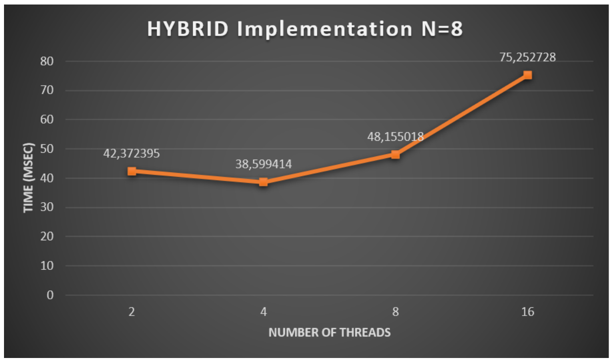

Fig. 8. Hybrid Implementation N=8

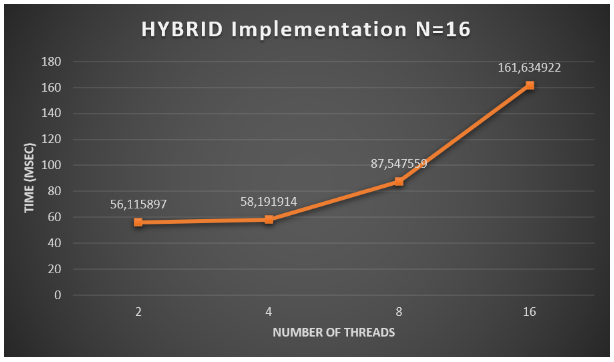

Fig. 9. Hybrid Implementation N=16

sions of our study below. Summary of our findings: MPI implementation: One of the main findings is that while MPI provides parallelism, there is a bottleneck when final image processing is assigned to the root rank. When dealing with many processes the cost with communicating with the root node exceeds the total processing time of the image resulting in an increased total execution time.OpenMP implementation: We found that OpenMP offers better performance than MPI. The more threads you added, the more efficient the implementation becomes. However, there are diminishing returns. Hybrid implementation: Our implementation, which combines MPI with OpenMP, shows similar results. It is important to note that the code implementations mentioned in our study are not fully optimized for the best possible performance. Therefore, further refinements and optimization could improve the performance of parallel Grayscale Image Processing. To sum up, our study shows that parallel grayscale Image Processing is feasible and provides valuable information on trade-offs and difficulties related to different parallel programming approaches. We stress the importance of selecting the correct parallel approach based on the specific task requirements and hardware resources available.

References

[1] Diniz, Wendell Horta, Agnus Nobrega, Euripedes Ferreira, Luiz. (2011). Parallel Implementation of Grayscale Conversion in Graphics Hardware.

[2] MMA, “RGB to Grayscale Conversion,” Mustafa Murat ARAT, May 13, 2020. https://mmuratarat.github.io/2020-05-13/rgb$_{to_g}rayscale_formulas$

[3] object Object, “Parallel Algorithms for Gray-Scale Digitized Picture Component Labeling on a Mesh-Connected Computer,” core.ac.uk, Accessed: Sep. 23, 2023. [Online]. Available: https://core.ac.uk/reader/4971640

[4] Saxena, Sanjay Sharma, Shiru Sharma, Neeraj. (2016). Parallel Image Processing Techniques, Benefits and Limitations. Research Journal of Applied Sciences, Engineering and Technology. 12. 223-238. 10.19026/rjaset.12.2324