OPERATING SYSTEMS LAB

CS 314 ASSIGNMENT 5

PRANAV KANIRE 190010033

ARUN R BHAT 190010007

1. IMAGE PROCESSING IN C++:

given image (converted to png)

1.1. RGB to Gray-scale Conversion:

The three different colors have three different wavelength and have their own contribution in the formation of image, so we have to take average according to their contribution, not done it averagely using average method. So we took weighted average of contributions from red, green and blue colors from each pixel and found the grayness value.

(source: *Tutorialspoint*)

Grayness value = (0.30 * R) + (0.59 * G) + (0.11 * B)

i.e., 30% red, 59% green and 11% blue

This is found to be the best weighted average which takes wavelength and soothing effect into consideration.

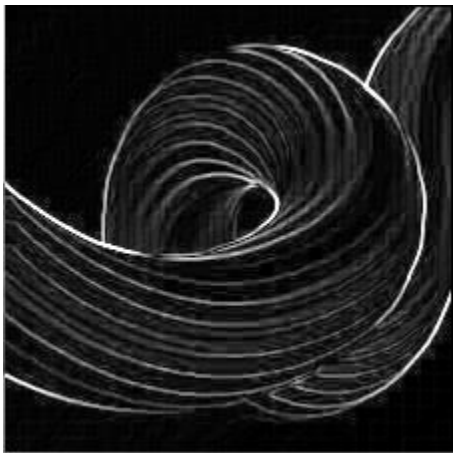

Image after T1 (in png)

1.2. Sobel Edge Detection:

In pictures, edges are areas with high intensity contrasts, or a jump in intensity from one pixel to the next. Edge detection reduces the amount of data by a large amount. It removes unnecessary data while keeping an image's crucial structural features. For edge detection, the Sobel filter is used. It calculates the image intensity gradient at each pixel inside the image. It determines the direction and pace of change in the most significant increase from light to dark. The result indicates how abruptly or smoothly the image changes at each pixel, and thus how probable that pixel is an edge.

Two 3 x 3 kernels are used in the Sobel filter. One for changes in the horizontal axis and the other for changes in the vertical axis. To calculate the estimates of the derivatives, the two kernels are convoluted with the original image. If we consider Gx and Gy to be two images containing the horizontal and vertical derivative approximations, the computations are as follows:

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

and where A is the original source image.

2. CORRECTNESS:

The operations applied are as follows: T1 * $[A]_{3x3}$ and T2 * $[B]_{3x3}$. Here $[A]_{3x3}$ and $[B]_{3x3}$ during time of operation should never have common elements, otherwise partially modified values by Tj will be picked up by Ti which will result in wrong output. In order to ensure that, this doesn't happen we have kept a size 2 array index_done = {0,0} available to both T1 and T2 functions as follows:

- In 2.1.a, it is global, since global variables are accessible by threads.
- In 2.1.b, it is global, since global variables are accessible by threads.
- In 2.2, it is pushed to a separate Shared Memory (shared memory in code) of (size(int) × 2) other than the pix map.
- In 2.3, it is pushed to separate pipe other than the pix map

3. RESULTS AND DISCUSSIONS:

3.1. Statistics:

Part	Method	input_1	input_2	input_3
1	Single core / simple	26839ms	31771ms	31202ms
2.1 a	Two threads - atomic	41958ms	44632ms	45104ms
2.1 b	Two threads - semaphores	54399ms	59617ms	66066ms
2.2	Two processes – shared memory - semaphores	67542ms	67293ms	6944ms
2.3	Two processes - pipes	74422ms	76687ms	77180ms

3.2. INFERENCE:

Executing using simple single core took the least amount of time.

Threading using atomic operations was faster than that using semaphores.

Executing two processes synchronized via shared memory was faster than that via piping.

3.3. CONCLUSION:

Though we thought using threading helps to improve the execution speed, we can observe here that it is not the case. This is due to the extra load / overhead on the machine to maintain synchronization between processes and also because of slight delaying caused in order to maintain the consistency of data. So we learn that this overhead has higher effect than the advantages of parallelism. This might be the result of smaller critical section where parallelism doesn't help much but only add the overheads.