INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS PUEBLA

MULTIPROCESADORES

GRUPO 1

PROF: EMANUEL TORRES

ACTIVIDAD 2.2

MULTI-THREADS

FERNANDO LÓPEZ RAMÍREZ - A07144620

PUEBLA, PUEBLA A 10 DE OCTUBRE DE 2022

INTRODUCCIÓN

El multithreading es el resultado de una interacción entre el hardware y el software. Los programas y procesos se dividen en hilos individuales y la CPU los procesa en estas unidades más pequeñas. Se distingue entre el multithreading basado en hardware y el basado en software.

En este caso realizamos diferentes tipos de ejercicios con programación en paralelo utilizando la librería OpenMP y usando nueva funcionalidades como for, schedule, collapse, sections, etc.

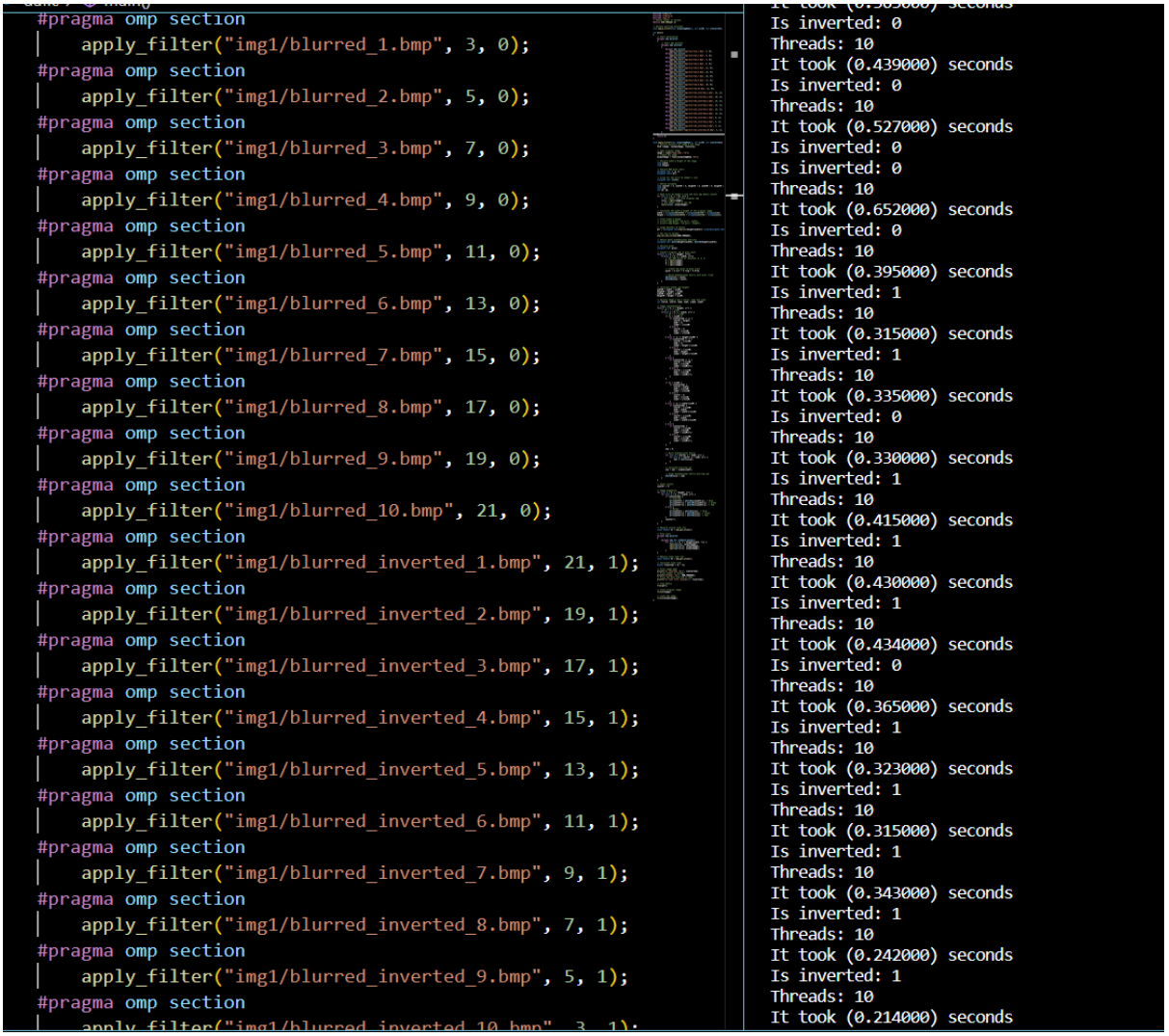
El lenguaje C nos permite en tiempo de ejecución solicitar espacio mediante la función malloc (memory allocate = Asignar memoria) y luego de usarla en forma obligada debemos devolverla llamando a la función free.

Estas dos funciones se encuentran en el archivo de inclusión: **#include<stdlib.h>**

DESARROLLO

Se realizó la actividad que cumple con los siguientes puntos imágenes (<700 píxeles por lado y >2000 pixeles por lado):

- Pasar imagen a escala de grises.
- Implementar paralelización.
- Girar horizontalmente la imagen.
- Hacer uso de memoria dinámica con **malloc.**
- Tiempo de ejecución con respecto a la tarea realizada sin el uso de threads.
- Generar 2 archivos .gif



```
#pragma omp section
    apply_filter("img1/blurred_1.bmp", 3, 0);
#pragma omp section
    apply_filter("img1/blurred_2.bmp", 5, 0);
#pragma omp section
    apply_filter("img1/blurred_3.bmp", 7, 0);
#pragma omp section
    apply_filter("img1/blurred_4.bmp", 9, 0);
#pragma omp section
    apply_filter("img1/blurred_5.bmp", 11, 0);
#pragma omp section
    apply_filter("img1/blurred_6.bmp", 13, 0);
#pragma omp section
    apply_filter("img1/blurred_7.bmp", 15, 0);
#pragma omp section
    apply_filter("img1/blurred_8.bmp", 17, 0);
#pragma omp section
    apply_filter("img1/blurred_9.bmp", 19, 0);
#pragma omp section
    apply_filter("img1/blurred_10.bmp", 21, 0);
#pragma omp section
    apply_filter("img1/blurred_inverted_1.bmp", 21, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_2.bmp", 19, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_3.bmp", 17, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_4.bmp", 15, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_5.bmp", 13, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_6.bmp", 11, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_7.bmp", 9, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_8.bmp", 7, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_9.bmp", 5, 1);
#pragma omp section
    apply_filter("img1/blurred_inverted_10.bmp", 3, 1);
```

```
Is inverted: 0
Threads: 10
It took (0.439000) seconds
Is inverted: 0
Threads: 10
It took (0.527000) seconds
Is inverted: 0
Is inverted: 0
Threads: 10
It took (0.652000) seconds
Is inverted: 0
Threads: 10
It took (0.395000) seconds
Is inverted: 1
Threads: 10
It took (0.315000) seconds
Is inverted: 1
Threads: 10
It took (0.335000) seconds
Is inverted: 0
Threads: 10
It took (0.330000) seconds
Is inverted: 1
Threads: 10
It took (0.415000) seconds
Is inverted: 1
Threads: 10
It took (0.430000) seconds
Is inverted: 1
Threads: 10
It took (0.434000) seconds
Is inverted: 0
Threads: 10
It took (0.365000) seconds
Is inverted: 1
Threads: 10
It took (0.323000) seconds
Is inverted: 1
Threads: 10
It took (0.315000) seconds
Is inverted: 1
Threads: 10
It took (0.343000) seconds
Is inverted: 1
Threads: 10
It took (0.242000) seconds
Is inverted: 1
Threads: 10
It took (0.214000) seconds
```

Imagen 1. Tiempos de ejecución para imagen de 1080 x 624 px: los primeros 10 son imágenes con el efecto blurring (Is Inverted = 0) y los 10 restantes son imágenes con el efecto blurring e invertidas (Is Inverted = 1)

OBSERVACIONES.

- Lamentablemente con la imagen de muestra de Mario Bros, la imagen se distorsiona cuando se intenta invertir, pero con la imagen de 1080 x 624 quedó muy bien.
- El GIF se adjuntará en la subida de archivos de la entrega.

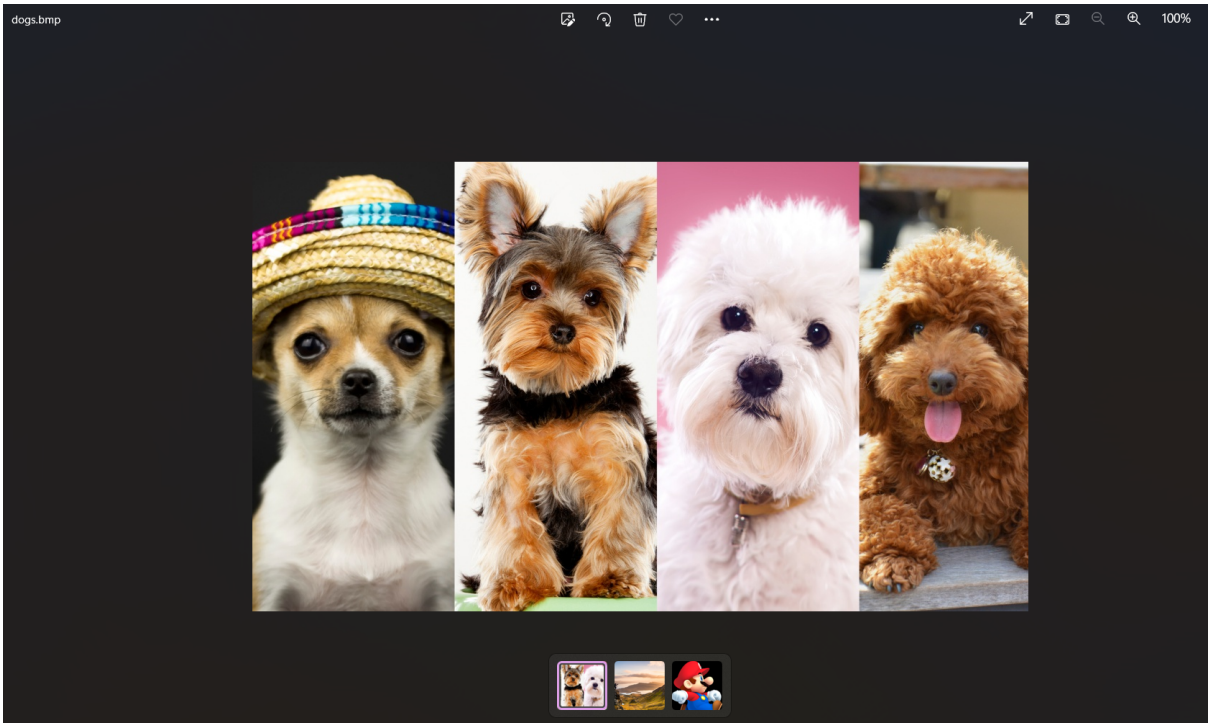


Imagen 2. Imagen original de 1080 x 624 px.

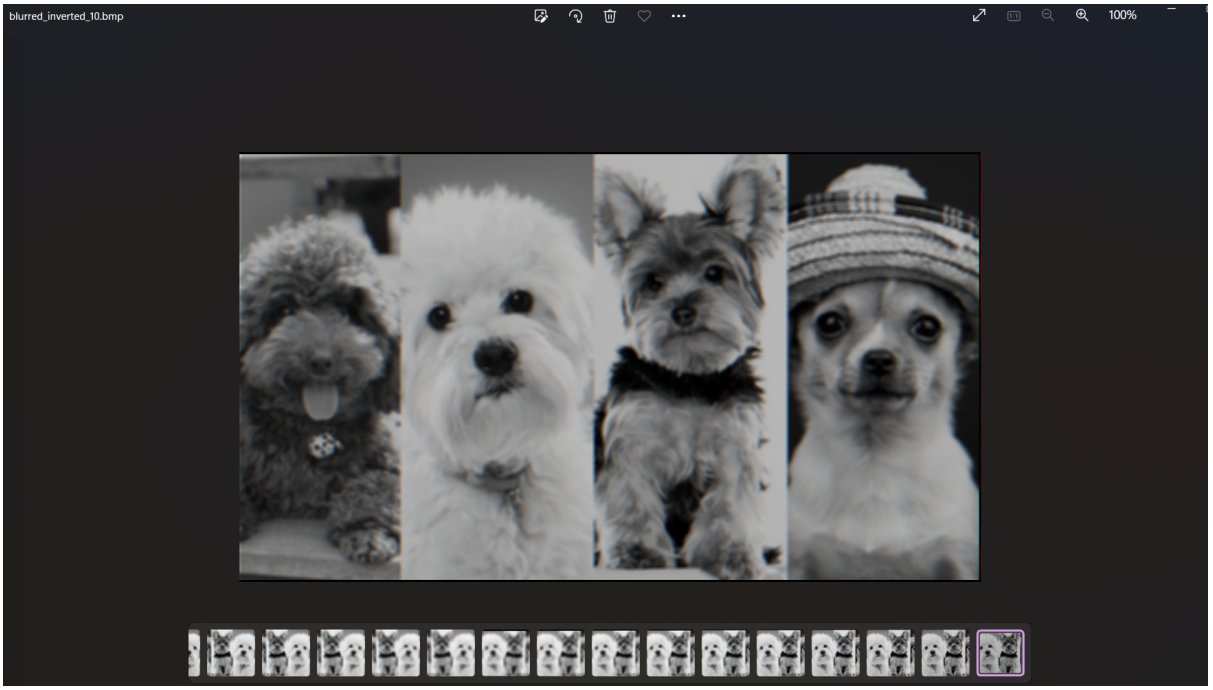


Imagen 3. Imagen en escala de grises de 1080 x 624 px y girada.

CÓDIGO.

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <omp.h>
// Define number of threads
#define NUM_THREADS 10

// Declare blurring functions
void apply_filter(char outputImgName[], int sizeM, int
isInverted);

int main()
{
    // Start parallelism
    #pragma omp parallel
    {
        // Start omp sections
        #pragma omp sections
        {
            #pragma omp section
                apply_filter("blurred_1.bmp", 3, 0);
            #pragma omp section
                apply_filter("blurred_2.bmp", 5, 0);
            #pragma omp section
                apply_filter("blurred_3.bmp", 7, 0);
            #pragma omp section
                apply_filter("blurred_4.bmp", 9, 0);
            #pragma omp section
                apply_filter("blurred_5.bmp", 11, 0);
            #pragma omp section
                apply_filter("blurred_6.bmp", 13, 0);
            #pragma omp section
                apply_filter("blurred_7.bmp", 15, 0);
            #pragma omp section
                apply_filter("blurred_8.bmp", 17, 0);
            #pragma omp section
                apply_filter("blurred_9.bmp", 19, 0);
            #pragma omp section
                apply_filter("blurred_10.bmp", 21, 0);
            #pragma omp section
                apply_filter("blurred_inverted_1.bmp", 21, 1);
            #pragma omp section
                apply_filter("blurred_inverted_2.bmp", 19, 1);
            #pragma omp section
```

```
                apply_filter("blurred_inverted_3.bmp", 17, 1);
            #pragma omp section
                apply_filter("blurred_inverted_4.bmp", 15, 1);
            #pragma omp section
                apply_filter("blurred_inverted_5.bmp", 13, 1);
            #pragma omp section
                apply_filter("blurred_inverted_6.bmp", 11, 1);
            #pragma omp section
                apply_filter("blurred_inverted_7.bmp", 9, 1);
            #pragma omp section
                apply_filter("blurred_inverted_8.bmp", 7, 1);
            #pragma omp section
                apply_filter("blurred_inverted_9.bmp", 5, 1);
            #pragma omp section
                apply_filter("blurred_inverted_10.bmp", 3, 1);
        }
    }
    return 0;
}

void apply_filter(char outputImgName[], int sizeM, int
isInverted){
    // Declare pointers for image & new image
    FILE *image, *outputImage, *lecturas;

    // Open original image
    image = fopen("dogs.bmp","rb");
    // Create new image
    outputImage = fopen(outputImgName,"wb");

    // Declare width & height of the image
    long width;
    long height;

    // Declare RGB pixel chars
    unsigned char r, g, b;
    unsigned char* ptr;

    // Array for the first 54 header's line
    unsigned char xx[54];
```

```
    // Define variables
    long counter = 0, widthR = 0, heightR = 0, widthM = 0,
heightM = 0;
    long sum;
    int iR, jR;

    // Read first 54 header's line and move img memory counter
    for (int i = 0; i < 54; i++) {
        // Get header line from original img
        xx[i] = fgetc(image);
        // Set header line to new img
        fputc(xx[i], outputImage);
    }

    // Calculate the width & height of the original image
    width = (long)xx[20]*65536 + (long)xx[19]*256 + (long)xx[18];
    height = (long)xx[24]*65536 + (long)xx[23]*256 +
(long)xx[22];

    // Print width & height
    // printf("Img Width: %ld px\n", width);
    // printf("Img Height: %ld px\n", height);

    // Asign pointers of malloc
    ptr = (unsigned char*)malloc(height*width*3* sizeof(unsigned
char));

    // Set num of threads
    omp_set_num_threads(NUM_THREADS);

    // Define photo bidimensional matrices
    unsigned char photo[height][width], photoB[height][width];

    // Declase pixel
    unsigned char pixel;

    // Convert original img to gray scale
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            // Get RGB values as received: B, G, R
            b = fgetc(image);
```

```
            g = fgetc(image);
            r = fgetc(image);

            // Create new pixel in gray scale
            pixel = 0.21*r + 0.72*g + 0.07*b;

            // Build bidimensional matrix with pixel triad
            photo[i][j] = pixel;
            photoB[i][j] = pixel;
        }
    }

    // Calculate widths and heights
    widthR = width / sizeM;
    heightR = height / sizeM;
    widthM = width % sizeM;
    heightM = height % sizeM;

    // Declare image X and Y starts, ends and sides
    int startX, startY, endX, endY, sideX, sideY;

    // Image representation
    for(int i = 0; i < height; i++) {
        // iR = i * sizeM;
        for(int j = 0; j < width; j++) {
            // jR = j * sizeM;
            if (i < sizeM) {
                if (isInverted == 1) {
                    startX = height;
                    endX = 0;
                    sideX = i+sizeM;
                } else {
                    startX = 0;
                    endX = i+sizeM;
                    sideX = i+sizeM;
                }
            } else if (i >= height-sizeM) {
                if (isInverted == 1) {
                    startX = i+sizeM;
                    endX = 0;
                    sideX = height-i+sizeM;
```

```
            } else {
                startX = i-sizeM;
                endX = height;
                sideX = height-i+sizeM;
            }
        } else {
            if (isInverted == 1) {
                startX = i-sizeM;
                endX = i+sizeM;
                sideX = sizeM*2+1;
            } else {
                startX = i-sizeM;
                endX = i+sizeM;
                sideX = sizeM*2+1;
            }
        }

        if (j < sizeM) {
            if (isInverted) {
                startY = width;
                endY = j+sizeM;
                sideY = j+sizeM;
            } else {
                startY = 0;
                endY = j+sizeM;
                sideY = j+sizeM;
            }
        } else if (j >= width-sizeM) {
            if (isInverted) {
                startY = j-sizeM;
                endY = width;
                sideY = width-j+sizeM;
            } else {
                startY = j-sizeM;
                endY = width;
                sideY = width-j+sizeM;
            }
        } else {
            if (isInverted) {
                startY = j-sizeM;
                endY = j+sizeM;
```

```
                    sideY = sizeM*2+1;
                } else {
                    startY = j-sizeM;
                    endY = j+sizeM;
                    sideY = sizeM*2+1;
                }
            }

            sum = 0;

            // Built bidimensional matrix
            for (int x = startX; x < endX; x++) {
                for (int y = startY; y < endY; y++) {
                    sum += photo[x][y];
                }
            }

            // Calculate blurring sum
            sum = sum / (sideX*sideY);

            // Asign bidimensional matrix blurring sum
            photoB[i][j] = sum;
        }
    }

    // Reset counter
    counter = 0;

    // Image asignation
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (isInverted) {
                // Inverted
                ptr[counter] = photoB[i][width-j]; // Blue
                ptr[counter+1] = photoB[i][width-j]; // Green
                ptr[counter+2] = photoB[i][width-j]; // Red
            } else {
                // Normal
                ptr[counter] = photoB[i][j]; // Blue
                ptr[counter+1] = photoB[i][j]; // Green
                ptr[counter+2] = photoB[i][j]; // Red
```

```
            }
            counter++;
        }
    }

    // Measure initial time (t1)
    const double t1 = omp_get_wtime();

    // Gray scale
    #pragma omp parallel
    {
        #pragma omp for schedule(dynamic)
            for (int i = 0; i < height*width; ++i) {
                fputc(ptr[i], outputImage);
                fputc(ptr[i+1], outputImage);
                fputc(ptr[i+2], outputImage);
            }
    }

    // Measure final time (t2)
    const double t2 = omp_get_wtime();

    // Calculate final time
    double finalTime = t2 - t1;

    // Print image name
    printf("Is inverted: %d\n", isInverted);
    // Print number of threads
    printf("Threads: %d\n", NUM_THREADS);
    // Print how much time did it take
    printf("It took (%lf) seconds\n", finalTime);

    // Free memory
    free(ptr);

    // Close original image
    fclose(image);

    // Close new image
    fclose(outputImage);
}
```

CONCLUSIÓN

Por el momento puedo concluir que haciendo uso de la librería OpenMP podemos utilizar mucha de la utilería que nos ofrece para optimizar procesos con paralelismo y concurrencia. También, el saber utilizar los apuntadores es muy importante ya que si no se hace un buen uso de ellos, podemos realizar un mal manejo de memoria. La optimización, filtros y calidad de una imagen manipulada depende mucho del manejo que hagamos de los píxeles y sus respectivos valores R, G y B.

REFERENCIAS

Moisset, D. (n.d.). Asignación dinámica de memoria (malloc y free). Tutorialesprogramacionya.com. Retrieved October 11, 2022, from https://www.tutorialesprogramacionya.com/cya/detalleconcepto.php?punto=37&codigo=37&inicio=30

Multithreading: más potencia para los procesadores. (n.d.). IONOS Digitalguide. Retrieved September 30, 2022, from https://www.ionos.mx/digitalguide/servidores/know-how/explicacion-del-multithreading/