Tecnológico de Monterrey

Iván Jesús Vázquez Rivera

Actividad 2.2

Multiprocesadores

Introducción

En esta actividad se espera utilizar imágenes en formato BMP (Windows Bitmap) de un tamaño pequeño (< 700 pixeles) y de gran tamaño (>2000 pixeles) y hacer el efecto de difumado (Blurring) y rotación en eje horizontal.

Se pretende generar 20 imágenes: 10 con el efecto de difuminado aumentando cada vez más y las otras 10 con el efecto aumentando paulatinamente pero con la rotación.

Se tiene que medir el tiempo generado por esta tarea con un programa secuencial y otro que utilice threads con OpenMP.

Implementación

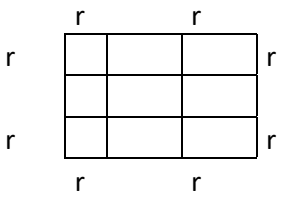
Para la implementación utilizamos la misma estructura que para la actividad 2.1: una estructura llamada pixel que contiene 3 unsigned char r, g, b para almacenar el valor de los registros.

Leemos el ancho y el alto del encabezado de la imagen haciendo una lectura de los registros.

Para hacer el efecto de difuminado tenemos que tomar los valores en un área cuadrada de los pixeles y obtener el promedio del valor de esos pixeles, posteriormente asignar ese valor en el pixel. Para ejemplificar la lógica del blurring hacemos el siguiente diagrama.

Pensemos en una venta de 3 x 3 y una imagen de 5 x 5

Aumento total = 2*r = 2 (1) = 2



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

r r

0	0	0	0	0	0	0	r
0	1	2	3	4	5	0	
0	6	7	8	9	10	0	
0	11	12	13	14	15	0	
0	16	17	18	19	20	0	
0	21	22	23	24	25	0	
0	0	0	0	0	0	0	r

r r

Para lograr el efecto es necesario aumentar la matriz considerando el tamaño de la ventana, en el ejemplo se usa una ventana de 3 x 3 por lo que aumentamos la matriz en uno de cada lado, en total dos, es importante mencionar que la ventana siempre es impar (para que pueda centrarse en un pixel) y el aumento a la matriz original es par.

Código implementado en C

```
void convolution(pixel **m, pixel **p, int width, int height, int widthC, int heightC){

    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            unsigned int value = 0;
            for (int x = 0; x < heightC; x++){
                for (int y = 0; y < widthC; y++){
                    value += p[i + x][j + y].b;
                }
            }
            m[i][j].b = value/(heightC*widthC);
            m[i][j].g = value/(heightC*widthC);
            m[i][j].r = value/(heightC*widthC);
        }
    }
}
```

Primero recorremos los valores de cada bit, al ser escala de grises cada valor de r,g,b es el mismo, por lo que no es necesario leer todos, sólo obtener el promedio y asignarlo a cada parte

Una vez teniendo la convolución lista reciclamos parte del código de la Actividad 2.1 para invertir en eje horizontal

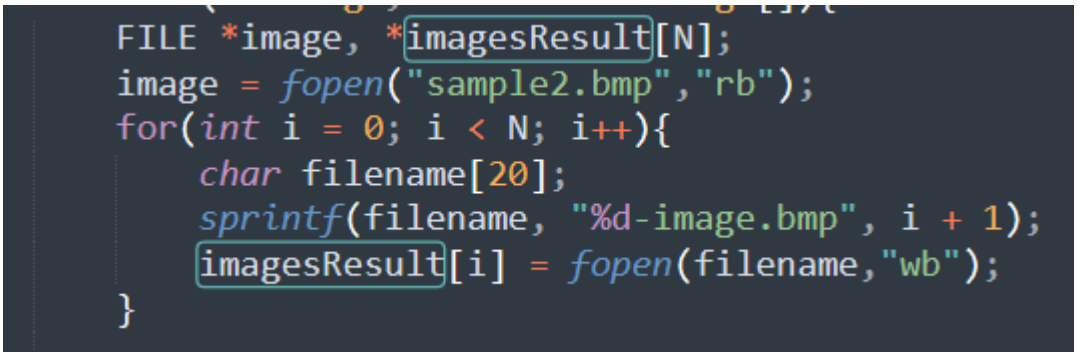
```
void changeHorizontal(pixel **m, int width, int height){
    pixel * array;
    array = (pixel*)malloc(sizeof(pixel)*width);

    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            array[j] = m[i][j];
        }

        for (int j = 0; j < width; j++){
            m[i][j] = array[width - j - 1];
        }
    }

    free(array);
}
```

Para generar los 20 archivos se utiliza un arreglo de archivos asignando un nombre diferente.



Se utiliza un mecanismo similar con fclose para cerrar los archivos. Para las pruebas usando la directiva sections section de OpenMP para que cada hilo genere una imagen diferente con un blur diferente.

Resultados

Comparamos los resultados de threads vs secuencial y estos fueron los resultados de rendimiento.

Imagen	Secuencial	Paralelismo
700 pixeles	4.1759 segundos	14.0992 segundos
2000 pixeles	1 minuto y 10 segundos	18.6874 segundos

Podemos ver que el uso de hilos mejora bastante el proceso de lectura, convolución, efecto espejo y escritura de las imágenes.

Se anexa resultados de las imágenes.