

Filtro de mediana implementado en C++ con manejo de hilos

Universidad Nacional de Colombia

Juan Jesus Pulido Sanchez
jjpulidos@unal.edu.co

Cristian Camilo Garcia Barrera
ccgarciaib@unal.edu.co

Danier Elian Gonzalez Ordóñez
dgonzalezo@unal.edu.co

I. DISEÑO

El filtro de mediana es un tipo de filtro no lineal comúnmente usado para eliminar el ruido de una imagen, este filtro funciona moviéndose a través de una imagen píxel a píxel, reemplazando cada valor con el valor de la mediana de los píxeles vecinos.

El patrón de píxeles vecinos se denomina como "ventana", la cual se desliza píxel a píxel sobre toda la imagen. La mediana se calcula ordenando primero todos los valores de píxeles de la ventana en orden numérico y luego se reemplaza el píxel que se está considerando con el valor del píxel medio.

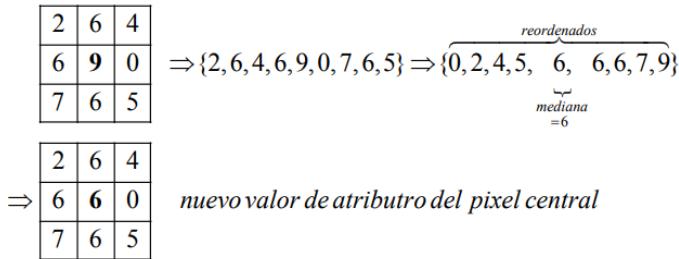


Figure 1. Ejemplo para un píxel y una ventana de 3x3.

Para el proceso de lectura y escritura de la imagen se hace uso de la biblioteca OpenCV, al momento de la lectura la función imread me permite cargar una imagen desde un archivo especificado que retorna una matriz y al momento de escritura me permite guardar una imagen en un archivo especificado a partir de una imagen en forma de matriz.

```
//Lectura de la imagen
Mat image=imread(filename, IMREAD_COLOR);
//Escritura de la imagen
imwrite(filename, image, compression_params)
```

Para la implementación secuencial se recorrió la matriz de la imagen por sus filas y columnas con un tamaño de ventana "ksize", reordenando la ventana y devolviendo su valor medio para cada iteración.

```
uchar medianFilterWindow(const cv::  
                           Mat &src, int i, int j){  
    vector<uchar> pixel(ksize * ksize);  
    for(int k = 0; k < ksize * ksize; ++k){  
        pixel[k]=src.at<uchar>(i+  
                               delta[k].first,  
                               j+ delta[k].second);  
    }  
    sort(pixel.begin(), pixel.end());  
    return pixel[(ksize * ksize) / 2];  
}  
  
Mat medianFilter(const cv::Mat &src){  
    dst = src.clone();  
    for(int i=1; i<src.rows-ksize; ++i){  
        for(int j=1; j< src.cols-ksize; ++j){  
            dst.at<uchar>(i, j) =  
                medianFilterWindow(src, i, j);  
        }  
    }  
    return dst;  
}
```

Para el proceso de paralelización se usa block-wise, de tal manera que se separa el problema entre el número de hilos. En este caso se separaron la cantidad de filas de la imagen entre el número de hilos, de forma que cada hilo calcula todos los píxeles de un conjunto consecutivo de filas.

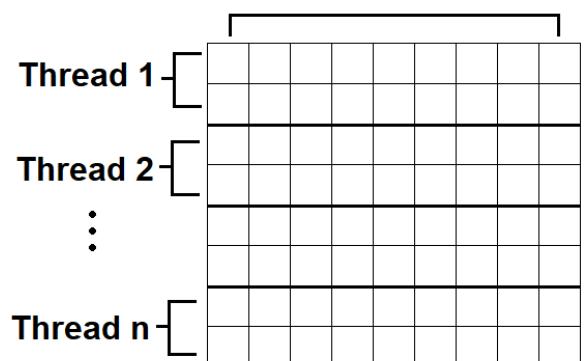


Figure 2. Ejemplo de funcionamiento de hilos.

Para la paraleización del código se realiza por medio de hilos POSIX con la librería pthread.h implementada en c++. Primero se inicializan arreglos para guardar los parámetros de ID's de los hilos:

```
//Thread variables
int threadId[total_threads];
pthread_t thread[total_threads];
```

Posteriormente se desarrolla un ciclo que creara todos los hilos. En este caso la función que se ejecuta se llama medianFilter.

```
for(int i = 0; i < total_threads; i++) {

    threadId[i] = i;
    pthread_create(&thread[i], NULL,
                  medianFilter, &threadId[i]);
}
```

Con los hilos creados se utiliza la función pthread_join() de tal manera que se espere hasta que todos los hilos terminen su ejecución.

```
//Join Threads
for(int i = 0; i < total_threads; i++) {
    pthread_join(thread[i], NULL);
}
```

II. PRUEBAS

En los experimentos se varian el numero de hilos (2, 4, 8, 16) y la calidad de las imágenes (720p, 1080p y 4K), con un valor de "ksize" estático en 5, sacando como resultado el promedio de 10 ejecuciones del programa con cada variación. La maquina usada para las pruebas cuenta con un procesador Intel i7-8750H con 12 hilos y 6 núcleos, con una capacidad de 16 GB de RAM.

Secuencial	
Calidad	Rt
720p	6,455721
1080p	15,03468
4k	59,66928

Figure 3. Resultado de tiempo de respuesta para la implementación secuencial

Imagen 720p	
#hilos	Rt
1	6,45572
2	3,32861
4	1,70501
8	1,17345
16	0,94793

Figure 4. Resultado de tiempo de respuesta para la implementación paralela con imagen 720p

Imagen 1080p	
#hilos	Rt
1	15,03468
2	7,67762
4	3,92419
8	2,70130
16	2,20903

Figure 5. Resultado de tiempo de respuesta para la implementación paralela con imagen 1080p

Imagen 4k	
#hilos	Rt
1	59,66928
2	30,47628
4	15,63322
8	12,37754
16	10,63257

Figure 6. Resultado de tiempo de respuesta para la implementación paralela con imagen 4k

Imagen 720p	
#hilos	SpeedUp
1	1,00000
2	1,93947
4	3,78634
8	5,50150
16	6,81034

Figure 7. Resultado de SpeedUp con imagen 720p

Imagen 1080p	
#hilos	SpeedUp
1	1,00000
2	1,95825
4	3,83128
8	5,56573
16	6,80600

Figure 8. Resultado de SpeedUp con imagen 1080p

Imagen 4k	
#hilos	SpeedUp
1	1,00000
2	1,95789
4	3,81683
8	4,82077
16	5,61193

Figure 9. Resultado de SpeedUp con imagen 4k



Figure 12. Imagen con ruido 720p

III. RESULTADOS



Figure 10. Comparación de tiempos de respuesta



Figure 13. Imagen después de aplicar filtro de mediana 720p

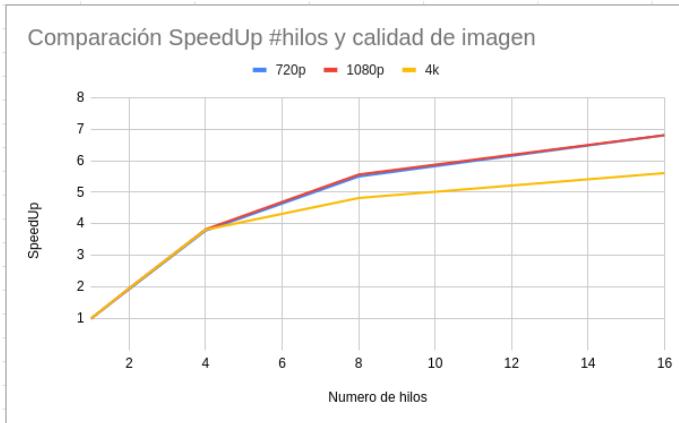


Figure 11. Comparación de SpeedUp

A continuación se hace una comparación de imágenes originales que tienen aplicado un filtro "sal y pimienta" y las imágenes resultado después de aplicar el filtro de mediana para reducir este ruido.



Figure 14. Imagen con ruido 4k



Figure 15. Imagen después de aplicar filtro de mediana 4k

IV. CONCLUSIONES

- La convolución como función matemática aplicada en imágenes es fácilmente paralelizable usando hilos POSIX usando C++, por lo tanto en caso de tener un proceso en el que se aplique dicho filtro a una cantidad de imágenes considerable, entonces podemos ahorrar costos computacionales al usar la implementación paralela del filtro de mediana.
- La mejora en velocidad gracias al paralelismo incrementa proporcionalmente al número de hilos después de 4 hilos.
- Dependiendo del tamaño de la imagen el aumento de velocidad de procesamiento cambia, como se evidencia que para imágenes muy grandes de resolución 4k el speed up tiende a ser menor que en imágenes 720p y 1080p después de 4 hilos.
- La paralelización permite acercar considerablemente los tiempos de respuesta para diferentes imágenes. Más investigación ha de ser realizada para llegar a una conclusión sólida, pero como hipótesis preliminar podría ser una mejora mayor en la localidad de la memoria procesada para imágenes más grandes.

V. REFERENCIAS

- Biblioteca OpenCV para la lectura y escritura de imágenes. Disponible en <https://opencv.org/>
- Marco teórico del filtro de mediana: http://fourier.eng.hmc.edu/e161/lectures/smooth_sharpen/node2.html