Efectoso Borroso de una imagen implementado en c con manejo de hilos

German David Guerrero Guerrero gdguerrerog@unal.edu.co, Ivan Camilo Quiroga Camargo
icquirogac@unal.edu.co, Jose David Salazar Moreno josdsalazarmor@unal.edu.co
Universidad Nacional de Colombia

I. Diseño

Por medio de la implementación un kernel gaussiano, implementado con la campana de Gauss para 2 dimensiones se calculan los valores de cada color en cada pixel de la nueva imagen:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

El tamaño de este kernel define la cantidad de pixeles al rededor del pixel objetivo que influyen en el valor de su color. Por ende entre mayor sea el tamaño del kernel, el efecto borroso sobre la imagen es mayor. Cabe añadir que hay un parametro adicional que define tambien la intensidad del efecto borroso y es el valor de σ en la ecuación del kernel. Este valor por defecto es de 10, y entre mayor sea, mayor sera la influencia que tendran los pixeles aledaños sobre el pixel objetivo.

```
// Pixel calc using kernel
// For each pixel in thread range
for(int i = iy; i <= ey && i < newImg->height; i++)
     for(int j = ix; j <= ex && j < newImg->width;
    j++){
    rvalue = gvalue = bvalue = sum = 0;
    // For each location in kernel
    for(int k = -1 * ksize/2; k <= ksize/2; k++) for
        (int l = -1 * ksize/2; l <= ksize/2; l++) {
        // Kernel location out of image
        if(i + k < 0 || i + k >= newImg->height || j
             + l < 0 || j + l >= newImg->width)
            continue;

        rvalue += img->pixels[i + k][j + l].R *
            kernel[k + ksize/2][l + ksize/2];
        gvalue += img->pixels[i + k][j + l].G *
            kernel[k + ksize/2][l + ksize/2];
        bvalue += img->pixels[i + k][j + l].B *
            kernel[k + ksize/2][l + ksize/2];

        // For normalization
        sum += kernel[k + ksize/2][l + ksize/2];
    }


    // Saving calculated values
    newImg->pixels[i][j].R = rvalue/sum;
    newImg->pixels[i][j].G = gvalue/sum;
    newImg->pixels[i][j].B = bvalue/sum;
}
```

Para el proceso de lectura de la imagen se usa la librería stb y una implementación para convertir esta lectura en pixeles en el archivo file_system.h, de tal manera que se pueda leer la imagen deseada y escribir la nueva imagen con el efecto borroso.

Para el proceso de asignación de memoria se usa malloc para asegurarse de la correcta asignación de memoria en la imagen trabajada, así como la nueva imagen resultante con el filtro borroso.

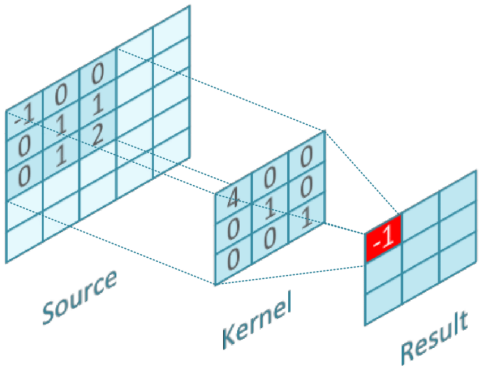
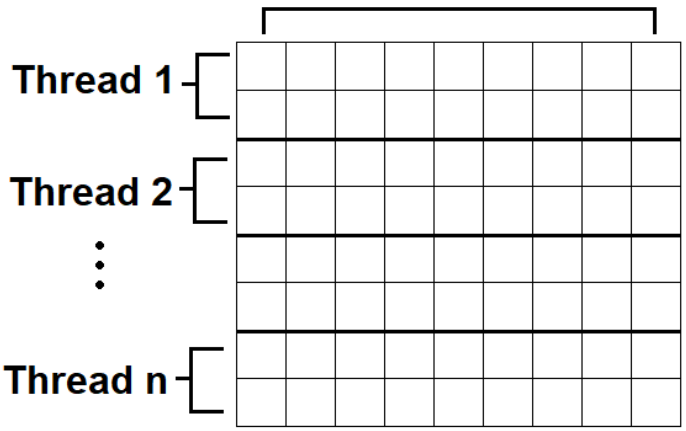


Fig. 1. Aplicación de un kernel sobre un pixel

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 pixeles de un conjunto consecutivo de filas.

Fig. 2. Ejemplo de funcionamiento de hilos



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

```
// Paralel Setup
pthread_t * thread_ids = (pthread_t *)malloc(
    threads * sizeof(pthread_t));
int * params = (int *)malloc(threads * sizeof(
    int));
```

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

```
// Thread Creation
for(int i = 0; i < threads; i++){
    printf("Launching thread %d\n", i);
    params[i] = i;
    pthread_create(&thread_ids[i], NULL, &blurCalc,
        &params[i]);
    printf("Launched thread %d with id %ld\n", i,
        thread_ids[i]);
}
```

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

```
// Thread Join
for(int i = 0; i < threads; i++){
    printf("Joining thread %d\n", i);
    pthread_join(thread_ids[i], NULL);
}
```

II. Pruebas

En los experimentos se probaron según el número de kernel asignado y según el número de hilos usados. La máquina usada para las pruebas tiene 4 núcleos físicos, y 8 procesadores lógicos. Además cuenta con 12GB de ram y utiliza un procesador Intel Core i7-6700HQ.

720	# Kernel						
	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
#Hilos 1	0,132964	0,173261	0,242048	0,333312	0,438314	0,571977	0,731592
2	0,111506	0,137466	0,172089	0,223504	0,278414	0,354298	0,428614
4	0,110539	0,136005	0,148709	0,180052	0,202217	0,292812	0,309231
8	0,111457	0,117883	0,137448	0,163597	0,189854	0,235356	0,280219
16	0,10361	0,121139	0,142172	0,1678	0,198539	0,247867	0,291305

Fig. 3. Resultados de tiempo en la imagen de 720p

1080	# Kernel						
	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
#Hilos 1	1,503763	2,146999	3,114961	4,335908	5,896086	7,772658	9,946646
2	1,319834	1,648083	2,176029	2,829434	3,640047	4,645329	5,781518
4	1,245307	1,424072	1,75812	2,006034	2,445791	2,980707	3,589169
8	1,227533	1,40189	1,661421	1,999453	2,462457	2,982528	3,613017
16	1,237455	1,401820	1,683401	2,021814	2,475940	3,004848	3,641905

Fig. 4. Resultados de tiempo en la imagen de 1080p

4k	# Kernel						
	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
#Hilos 1	6,230705	8,783415	12,557868	17,512574	23,721756	31,187608	39,966445
2	5,575067	6,88542	8,861975	11,473304	14,782308	18,746672	23,401473
4	5,209112	5,847527	6,995248	8,311347	10,013782	12,055583	14,477256
8	5,208371	5,87517	6,948741	8,301229	10,054341	12,135409	14,576287
16	5,220075	5,883938	6,926169	8,277415	10,059966	12,134995	14,616238

Fig. 5. Resultados de tiempo en la imagen de 4K

	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
1	1	1	1	1	1	1	1
2	1,19243808	1,26039166	1,40652802	1,49130217	1,57432457	1,61439523	1,70687845
4	1,20286958	1,27393111	1,62766208	1,85119854	2,16754279	1,9533933	2,36584301
8	1,19296231	1,46977087	1,76101507	2,03739677	2,30868984	2,43026309	2,61078656
16	1,28331242	1,43026606	1,7025012	1,98636472	2,20769723	2,30759641	2,5114296

Fig. 6. Resultados de speedup en la imagen de 720p

Speedup							
	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
1	1	1	1	1	1	1	1
2	1,13935768	1,30272504	1,43148873	1,53242945	1,61978293	1,67322013	1,72042118
4	1,207544	1,50764779	1,77175676	2,16143296	2,41070721	2,60765583	2,77129497
8	1,22502857	1,53150318	1,87487759	2,1685471	2,39439146	2,60606372	2,75300282
16	1,21520621	1,53157966	1,8503975	2,14456325	2,38135254	2,58670588	2,73116569

Fig. 7. Resultados de speedup en la imagen de 1080p

Speedup							
	Kernel=3	Kernel=5	Kernel=7	Kernel=9	Kernel=11	Kernel=13	Kernel=15
1	1	1	1	1	1	1	1
2	1,11760182	1,27565421	1,41705071	1,52637584	1,60473967	1,66363438	1,70786023
4	1,19611654	1,50207344	1,79519983	2,10706808	2,36891077	2,58698464	2,76063675
8	1,19628671	1,4950061	1,80721486	2,1096363	2,35935463	2,5699676	2,74188104
16	1,1936045	1,49277831	1,81310447	2,11570569	2,3580354	2,57005528	2,73438658

Fig. 8. Resultados de speedup en la imagen de 4K

III. Resultados

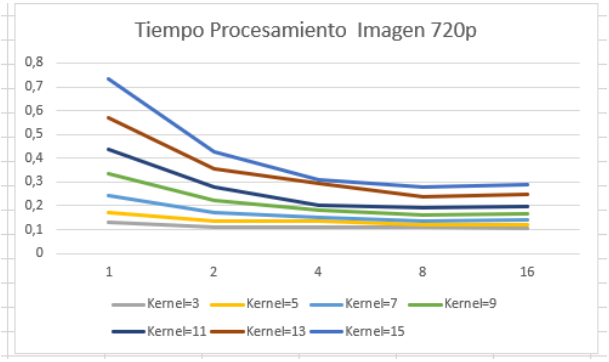


Fig. 9. Gráficas de las pruebas de tiempo en la imagen de 720p en función del número de hilos

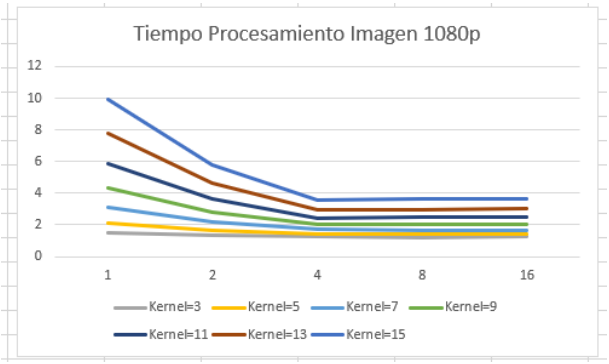


Fig. 10. Gráficas de las pruebas de tiempo en la imagen de 1080p en función del número de hilos

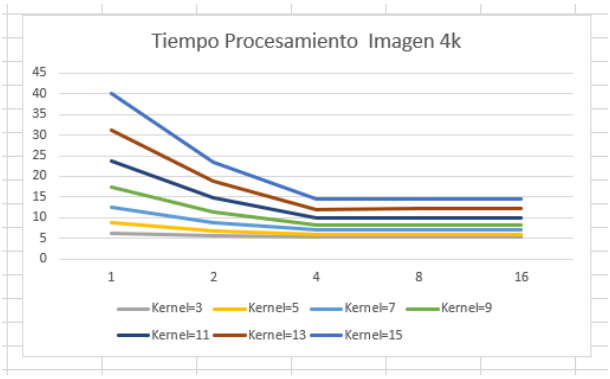


Fig. 11. Graficas de las pruebas de tiempo en la imagen de 4K en función del número de hilos

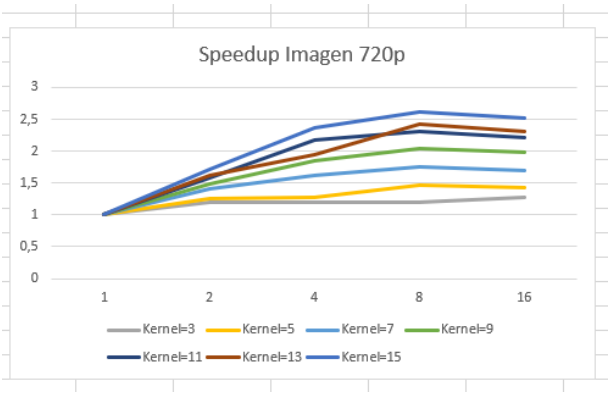


Fig. 12. Gráficas de las pruebas de speedup en la imagen de 720p en función del número de hilos

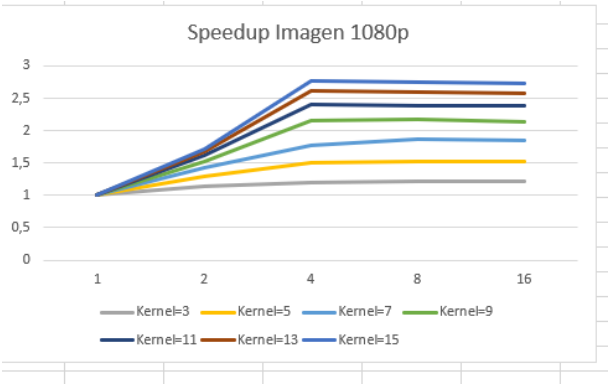


Fig. 13. Gráficas de las pruebas de speedup en la imagen de 1080p en función del número de hilos

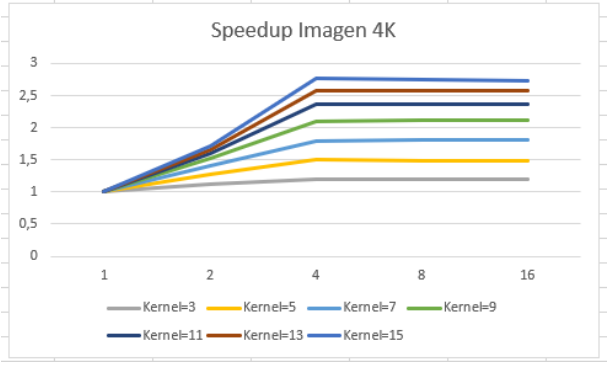


Fig. 14. Gráficas de las pruebas de speedup en la imagen de 4K en función del número de hilos

IV. Conclusiones

- Entre mas grande es el tamaño del kernel, el speed-up obtenido es mayor
- Despues de 4 hilos, no es significativo el aumento en el rendimiento del algoritmo. Problamente esto se deba a que la máquina usada solo tiene 4 núcleos físicos. Al parecer el hecho de que la máquina tenga 8 hilos realmente no mejora el rendimiento.
- El máximo speed-up que se puede obtener con la máquina usada es de aproximadamente 2.7

V. Referencia

- Librería STB para lectura de imágenes. Disponible en https://github.com/nothings/stb
- Como implementar un Gaussian Blur. Disponible en https://computergraphics.stackexchange.com/questions/39/how-is-gaussian-blur-implemented
- Como implementar un Gaussian Kernel. Disponible en https://stackoverflow.com/questions/8204645/implementing-gaussian-blur-how-to-calculate-convolution-matrix-kernel
- Figura 1 tomada de https://www.researchgate.net/figure/A-valid-convolution-of-a-5x5-image-with-a-3x3-kernel-The-kernel-will-be-applied-to_fig5_322505397