Computación Paralela y Distribuida
Práctica 1

Vinnie Giuliano Mellizo Molina, Julio Cesar Ovalle Lara.

Bogotá
Universidad nacional de Colombia

vgmellizom@unal.edu.co, jcovallel@unal.edu.co.

Resumen-
El presente documento constituye un reporte sobre la implementación y paralelización en C de un algoritmo para aplicar el efecto borroso sobre imágenes de diferentes tamaños. El objetivo de este reporte es ver el impacto de la paralelización y el impacto de otras variables como el tamaño de la imagen o el tamaño del kernel sobre los tiempos de ejecución del programa. Se espera ver el Speedup obtenido para cada situación.

Palabras Clave – *Paralelización – Efecto Borroso - Kernel - Tiempo de ejecución - Speedup.*

I. MARCO TEÓRICO

1.1 *Efecto borroso o desenfoque:*

Cuando se desenfoca una imagen, la transición de color desde un borde de la imagen a otro se torna más suave, esto puede ser útil para reducir el ruido en una imagen o para realizar otro tipo de tareas como la detección de bordes. El efecto de desenfoque o efecto borroso consiste básicamente en promediar la intensidad de los píxeles a lo largo de toda la imagen obteniendo así una imagen mucho más suave y homogénea. El desenfoque o suavizado de una imagen elimina los píxeles "atípicos" que pueden ser ruido, aunque deja intacta la mayor parte de la imagen. El desenfoque es un ejemplo de aplicación de un filtro de paso bajo a una imagen, en computación visual, el término "filtro de paso bajo" se refiere a la eliminación del ruido en una imagen [1].

1.2 *Gaussian blur:*

La convolución de dos funciones f y g en dos dimensiones se define como el volumen del producto de f y g "desplazado". A la segunda función g a veces se le llama "peso", ya que determina cuánto de f entrará en el resultado. Para el caso de las imágenes se utiliza la forma discreta de *Gaussian blur*, las funciones se representan como matrices y el cálculo del volumen (la integral) se calcula como una suma. Una función Gaussiana tiene valores cercanos a cero después de un determinado radio de manera que solo se usan los valores dentro de un radio especifico $-r \leq x \leq r, -r \leq y \leq r$. Al área delimitada dentro de este radio se le conoce como **kernel.** El valor de la convolución en la posición [i, j] es el promedio ponderado, es decir la suma de los valores de la función alrededor de [i, j] multiplicada por un peso [2].

$$b[i,j] = \sum_{y=i-r}^{i+r} \sum_{x=j-r}^{j+r} f[y,x] * w[y,x]$$

Figura 1. Gaussian blur para una matriz genérica en la posición [i, j].

1.3 *Box blur:*

Se define como la convolución de una función f y un peso w, pero el peso es constante y se encuentra dentro de una región cuadrada o con forma de caja de allí su nombre. La propiedad más interesante de Box blur es que el resultado de múltiples repeticiones (convoluciones) de Box blur se aproximan bastante a una convolución de Gaussian Blur. Sea br el radio de la caja y $1/(2 \cdot br)^2$ el valor constante del peso, entonces se puede calcular Box blur en cualquier punto [i, j] de la manera en que se muestra en la figura 2 [2].

$$bb[i,j] = \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br)^2$$

Figura 2. Box blur para una matriz genérica en la posición [i, j]

1.3 *Box blur optimizado:*

Se pueden establecer mejoras sobre el cálculo de Box blur, para la primera de estas mejoras se define un **blur horizontal** y un **blur total** ver Figura 3 [2].

$$b_h[i,j] = \sum_{x=j-br}^{j+br} f[i,x]/(2 \cdot br)$$

$$b_t[i,j] = \sum_{y=j-br}^{j+br} b_h[y,j]/(2 \cdot br)$$

Figura 3. Blur horizontal y blur total

Ambas funciones están iterando sobre una línea, produciendo así un "blur de una sola dimensión" sin embargo la Figura 4 muestra un hecho interesante sobre el blur total [2].

$$b_t[i,j] = \sum_{y=i-br}^{i+br} b_h[y,j]/(2 \cdot br)$$

$$= \sum_{y=j-br}^{j+br} \left(\sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br) \right) /(2 \cdot br)$$

$$= \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br)^2$$

Figura 4. Desarrollo del blur total

Y es que el blur total es equivalente al cálculo original de Box blur Sin embargo tanto el blur total como el horizontal tienen una complejidad $O(n \cdot r)$ mientras que el cálculo de box blur original tiene una complejidad $O(n \cdot r^2)$.

Finalmente el cálculo del blur unidimensional se puede hacer aún más rápido si se tiene en cuenta que al calcular ***bh*[*i*,*j*]**, ***bh*[*i*,*j*+1]**, ***bh*[*i*,*j*+2]**, **....** los valores vecinos ***bh*[*i*,*j*]** y ***bh*[*i*,*j*+1]** son casi iguales. La única diferencia es en el valor de más a la izquierda y el valor de más a la derecha de sus kernel. De manera que se puede optimizar su cálculo si se tiene en cuenta la igualdad mostrada en la Figura 5 [2].

$$b_h[i,j+1] = b_h[i,j] + f[i,j+r+1] - f[i,j-r].$$

Figura 5. Calculo optimizado del blur horizontal.

II. DESARRROLLO

Para el desarrollo de la práctica se implementó Box blur con las optimizaciones presentadas en la sección anterior con el fin de reducir la complejidad de su cálculo. Para facilitar el manejo de las imágenes se usó la librería FreeImage la cual cuenta con una gran variedad de funciones para realizar operaciones de lectura y escritura sobre las imágenes.

La implementación de Box blur se realizó mediante una función (BlurFunc2) la cual se encarga de recorrer la imagen píxel por píxel calculando el promedio dentro del kernel y finalmente genera la imagen con el efecto borroso.

Para la paralelización de este cálculo se usó un balanceo de carga de tipo block-wise ya que se parte la imagen en tantos fragmentos como hilos haya, cada uno de los hilos lanzados ejecuta la función descrita anteriormente con el fragmento de imagen que le corresponda a tal hilo. Al final la conjunción de estos fragmentos genera el efecto borroso sobre la totalidad de la imagen.

III. ANÁLISIS Y RESULTADOS.

Para la obtención de resultados y el posterior análisis de estos, se elaboró un script el cual se encarga de ejecutar el programa con diferentes imágenes, diferente tamaño de kernel y diferente número de hilos.

Los datos que se muestran a continuación fueron tomados sobre una maquina con procesador Intel Core i7-7700 el cual cuenta con 4 núcleos y 8 hilos, también se realizaron pruebas en 3 máquinas más con diferentes procesadores sin embargo los datos de estas pruebas no se encuentran consignados en este documento. Las configuraciones realizadas y los resultados obtenidos se muestran a continuación:

3.1 *Imagen 720p*

1 hilo	
Kernel	**Tiempo (s)**
3	**0.078**
5	**0.071**
7	**0.088**
9	**0.087**
11	**0.096**

13	**0.069**
15	**0.073**

2 hilos	
Kernel	**Tiempo (s)**
3	**0.066**
5	**0.062**
7	**0.079**
9	**0.077**
11	**0.088**
13	**0.067**
15	**0.065**

4 hilos	
Kernel	**Tiempo (s)**
3	**0.069**
5	**0.064**
7	**0.097**
9	**0.082**
11	**0.092**
13	**0.061**
15	**0.067**

8 hilos	
Kernel	**Tiempo (s)**
3	**0.084**
5	**0.085**
7	**0.1**
9	**0.102**
11	**0.104**
13	**0.079**
15	**0.084**

16 hilos	
Kernel	**Tiempo (s)**
3	**0.109**
5	**0.111**
7	**0.122**
9	**0.121**
11	**0.133**
13	**0.117**
15	**0.106**

3.2 *Imagen 1080p*

1 hilo	
Kernel	**Tiempo (s)**
3	**0.144**
5	**0.142**
7	**0.144**
9	**0.143**
11	**0.141**
13	**0.144**
15	**0.141**

2 hilos	
Kernel	**Tiempo (s)**
3	**0.12**
5	**0.119**
7	**0.121**
9	**0.117**
11	**0.117**
13	**0.119**
15	**0.119**

4 hilos	
Kernel	**Tiempo (s)**
3	**0.137**
5	**0.12**
7	**0.121**
9	**0.12**

11	**0.121**
13	**0.125**
15	**0.123**

8 hilos	
Kernel	**Tiempo (s)**
3	**0.159**
5	**0.153**
7	**0.15**
9	**0.155**
11	**0.15**
13	**0.153**
15	**0.156**

16 hilos	
Kernel	**Tiempo (s)**
3	**0.207**
5	**0.209**
7	**0.208**
9	**0.205**
11	**0.208**
13	**0.204**
15	**0.215**

3.3 ***Imagen 4k***

1 hilo	
Kernel	**Tiempo (s)**
3	**1.457**
5	**1.417**
7	**1.467**
9	**1.429**
11	**1.454**
13	**1.412**
15	**1.411**

2 hilos	
Kernel	**Tiempo (s)**
3	**1.172**
5	**1.182**
7	**1.193**
9	**1.158**
11	**1.153**
13	**1.184**
15	**1.175**

4 hilos	
Kernel	**Tiempo (s)**
3	**1.152**
5	**1.185**
7	**1.152**
9	**1.206**
11	**1.157**
13	**1.185**
15	**1.165**

8 hilos	
Kernel	**Tiempo (s)**
3	**1.559**
5	**1.559**
7	**1.554**
9	**1.565**
11	**1.564**
13	**1.563**
15	**1.548**

16 hilos	
Kernel	**Tiempo (s)**
3	**2.136**
5	**2.119**
7	**2.158**

9	**2.147**
11	**2.142**
13	**2.078**
15	**2.134**

El análisis de los datos recolectados en las anteriores tablas muestra varios hechos que pueden verse de manera más clara con la ayuda de las gráficas de estos datos, dichas graficas pueden ser consultadas en los anexos de este documento. Como se mencionó al principio de esta sección, aunque los datos mostrados aquí corresponden únicamente a una sola máquina, también se realizaron algunas pruebas sobre 3 máquinas más, una de ellas con procesador Intel core i5-3230M dos núcleos cuatro hilos, otra con procesador ryzen 5 3600 de seis núcleos 12 hilos y una máquina virtual de Google con 16 núcleos. De estas pruebas se pudo notar un hecho interesante y es que después de que se alcanzaba un numero de hilos igual al número de núcleos del procesador el tiempo que tardaba el programa en finalizar empezaba a aumentar. Para el caso del core i5 esto ocurría al lanzar 2 hilos, para el caso del ryzen 5 y el core i7 esto ocurría después de los 4 hilos y para el caso de la máquina virtual de Google esto ocurría después de 8 hilos. Algunas capturas de pantalla de los datos obtenidos en estas pruebas pueden ser encontradas en los anexos de este documento.

Otro hecho interesante que se puede notar fácilmente en las gráficas es que conforme el tamaño de la imagen crece la diferencia de tiempo entre los diferentes tamaños de kernel empieza a ser cada vez menor, es decir que realizar los cálculos para un kernel grande toma un tiempo muy similar a cuando se calcula con un kernel mucho más pequeño.

IV. CONCLUSIONES.

- Antes de pensar en paralelizar es importante optimizar el programa secuencial. Si bien el speedup máximo alcanzado en nuestras pruebas fue de 1.2, el algoritmo sin optimizar con O(k*k*h*w) representaría tiempos de ejecución demasiado elevados para los peores casos, por ejemplo, imágenes y/o kernel muy grandes.

- De acuerdo con la ley de Amdahl, el speedup obtenido apunta a un porcentaje de paralelización menor al 50% en nuestro programa, lo que podría deberse a los altos tiempos de carga y clonación de imágenes en contraste con las rápidas operaciones aritméticas sobre el bitmap de la imagen.

- Parece existir una relación directa entre el número de núcleos físicos y el desempeño de los hilos, ya que en nuestras pruebas el speedup comenzaba a decrecer una vez se superaba el número de núcleos físicos y no de hilos.

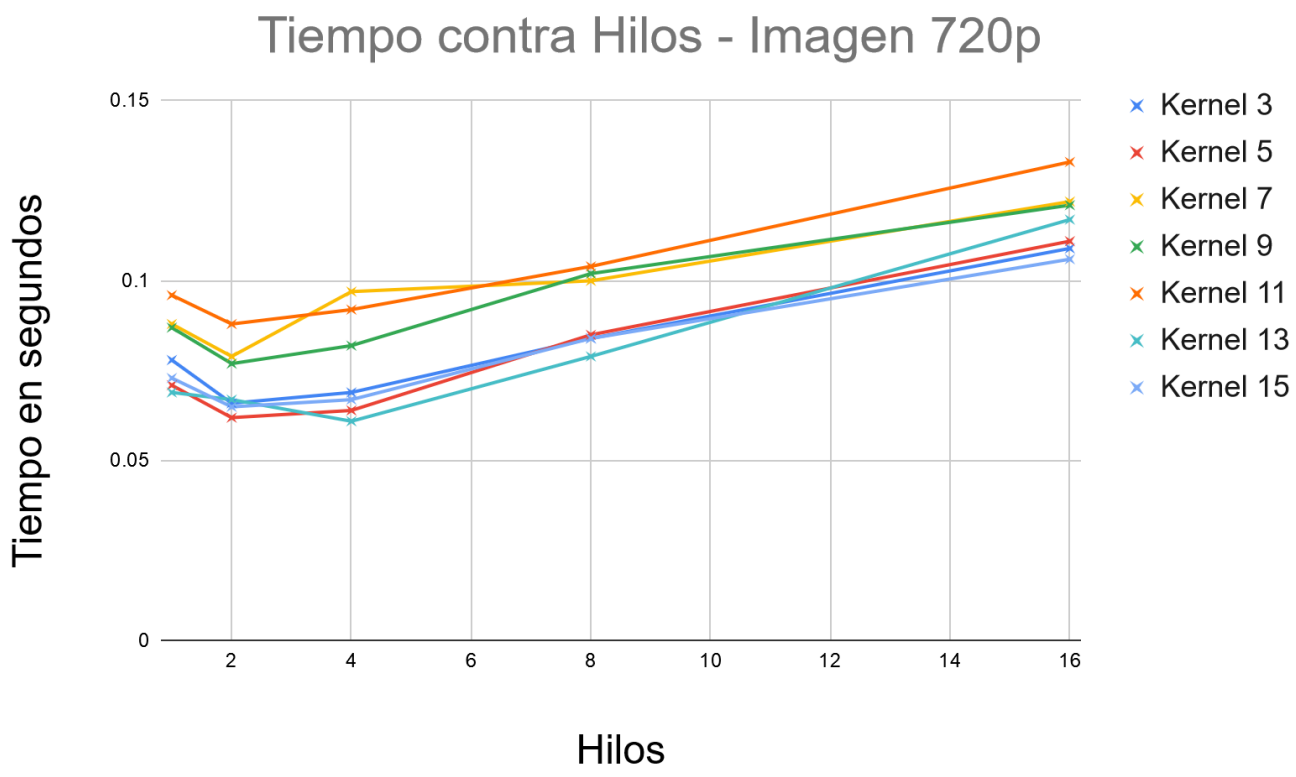
- El Box Blur es una alternativa rápida, fácil de implementar y de calidad suficiente en contraste con el Gaussian Blur que en la mayoría de los casos resultaría inviable de utilizar por su alta carga computacional.
- No siempre más núcleos representan mayor rapidez, existe un umbral a partir del cual, el tiempo extra de balancear la cagar, gestionar y comunicar los hilos excede el tiempo de ejecución ahorrado al paralelizar el programa. Todo dependerá del tamaño de los datos, su nivel de optimización y la máquina en la que se ejecuta.

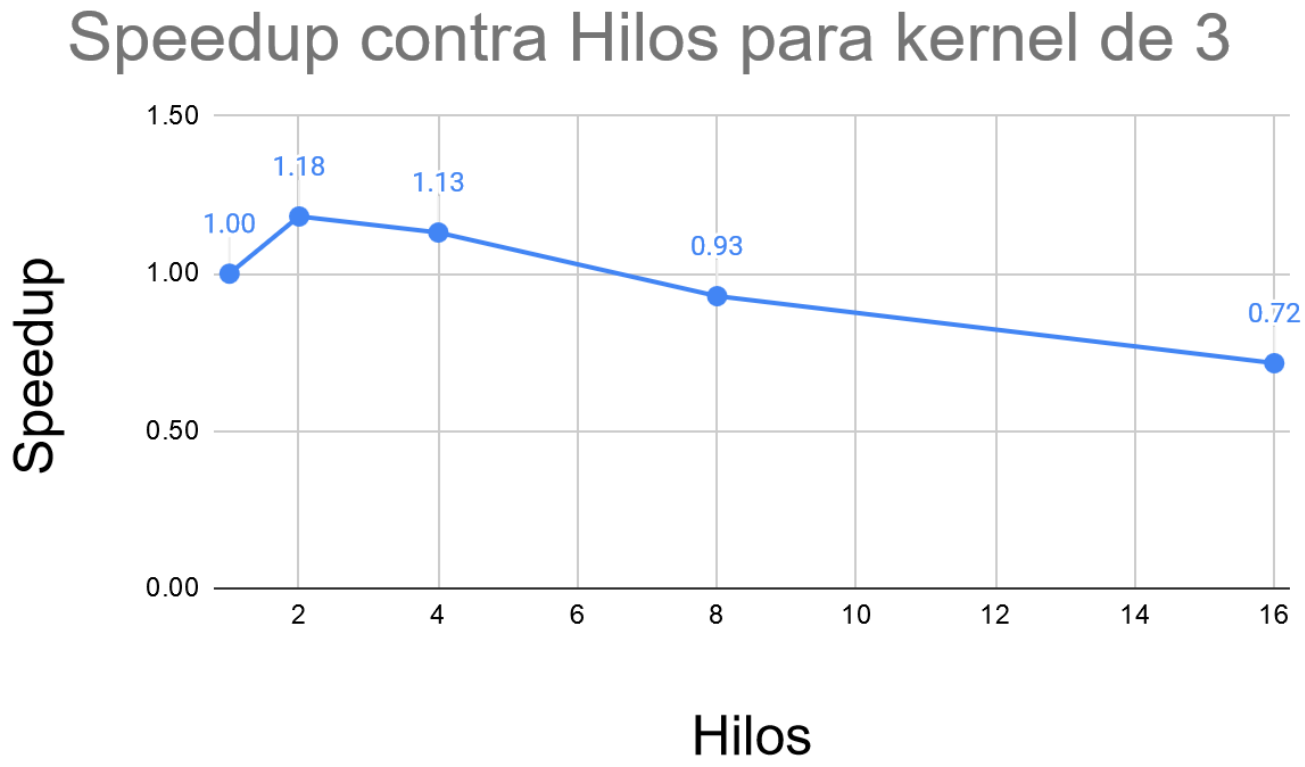
BIBLIOGRAFÍA

- [1] https://datacarpentry.org/image-processing/06-blurring/

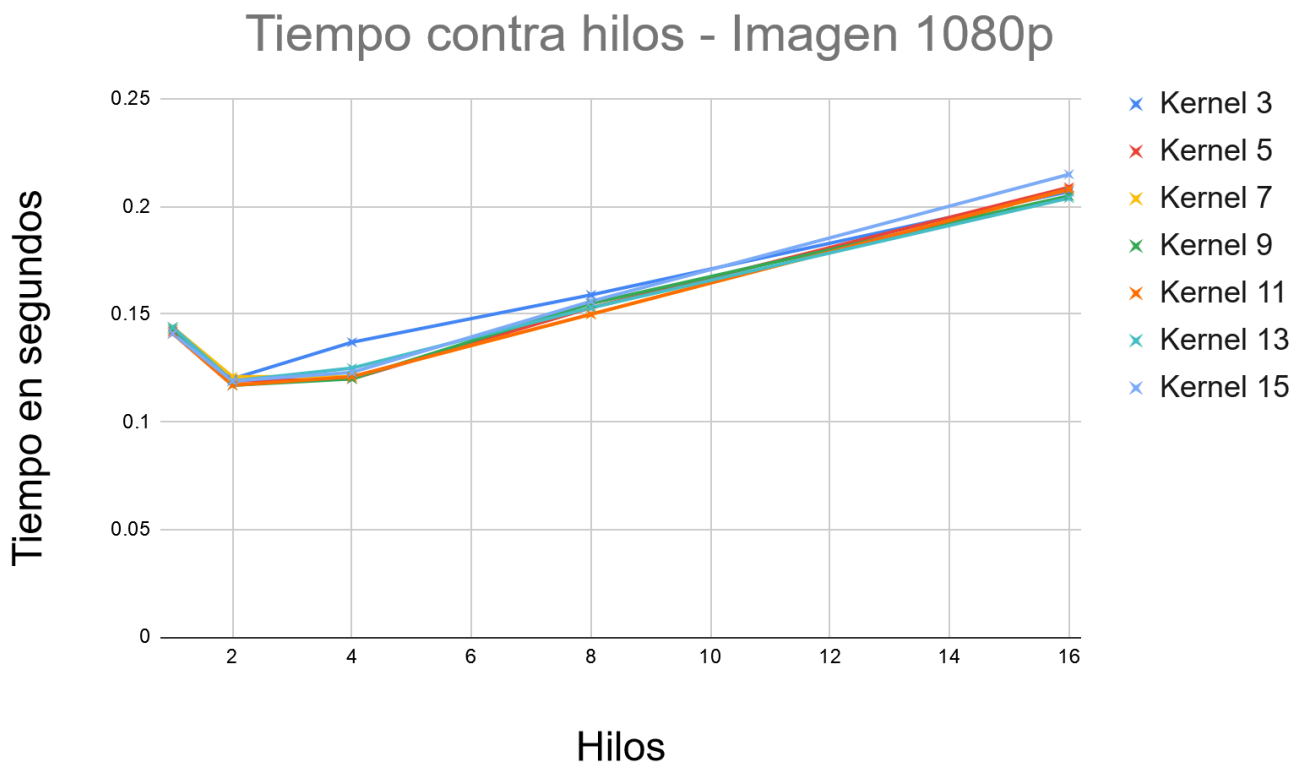
- [2] http://blog.ivank.net/fastest-gaussian-blur.html

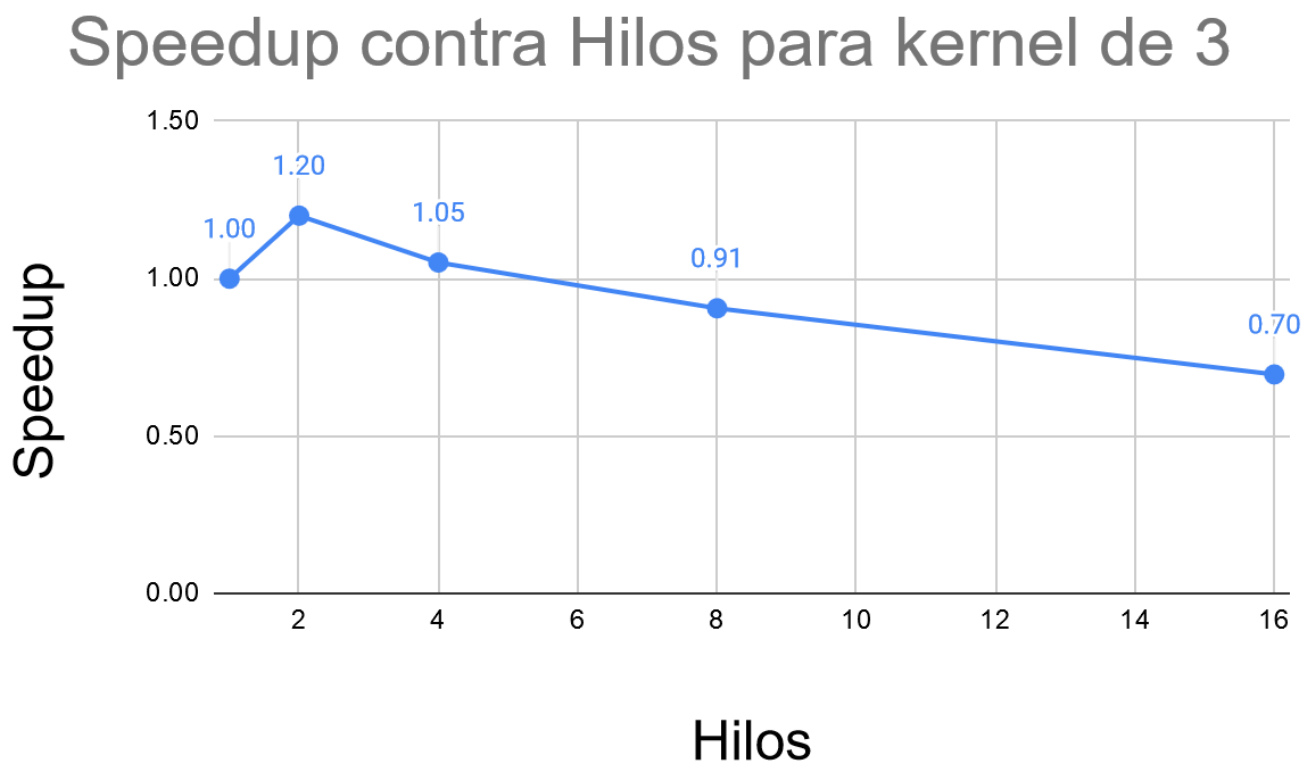
Anexo A
Graficas hilos vs tiempo y Speedup para imagen 720p



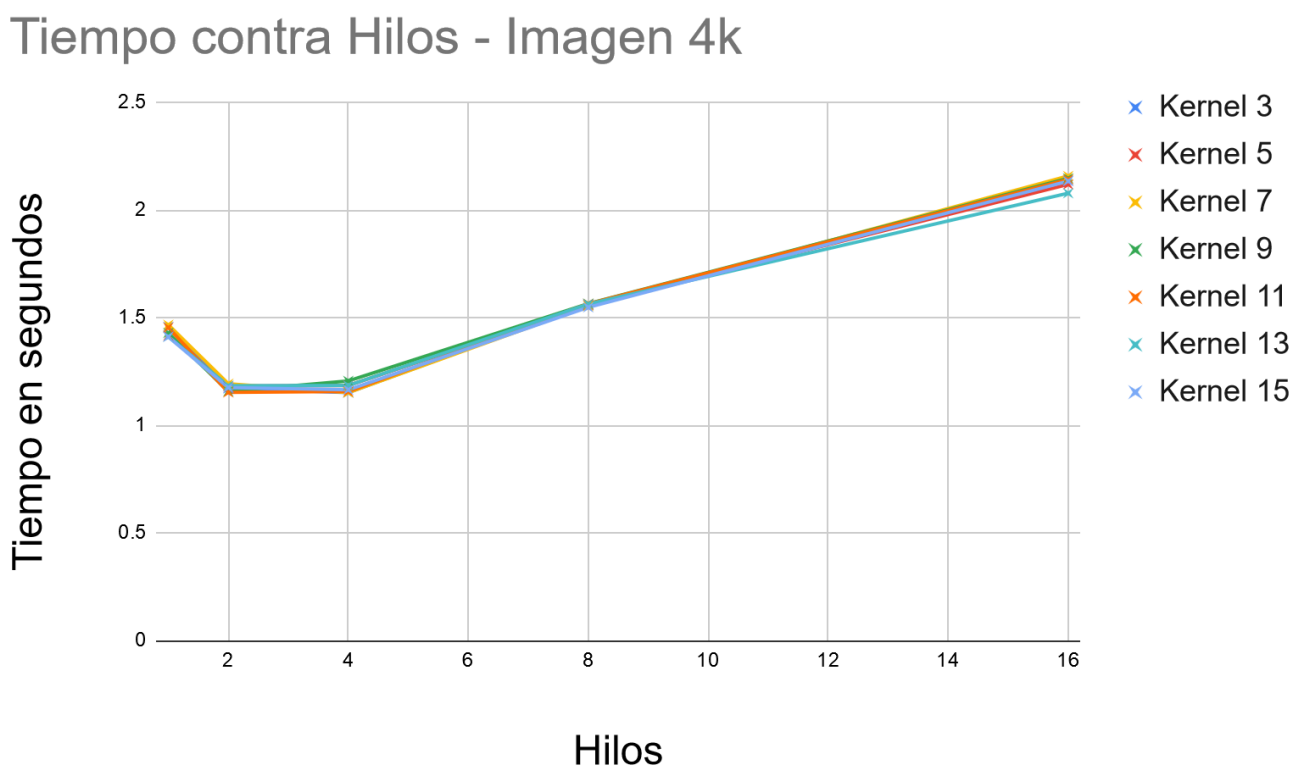


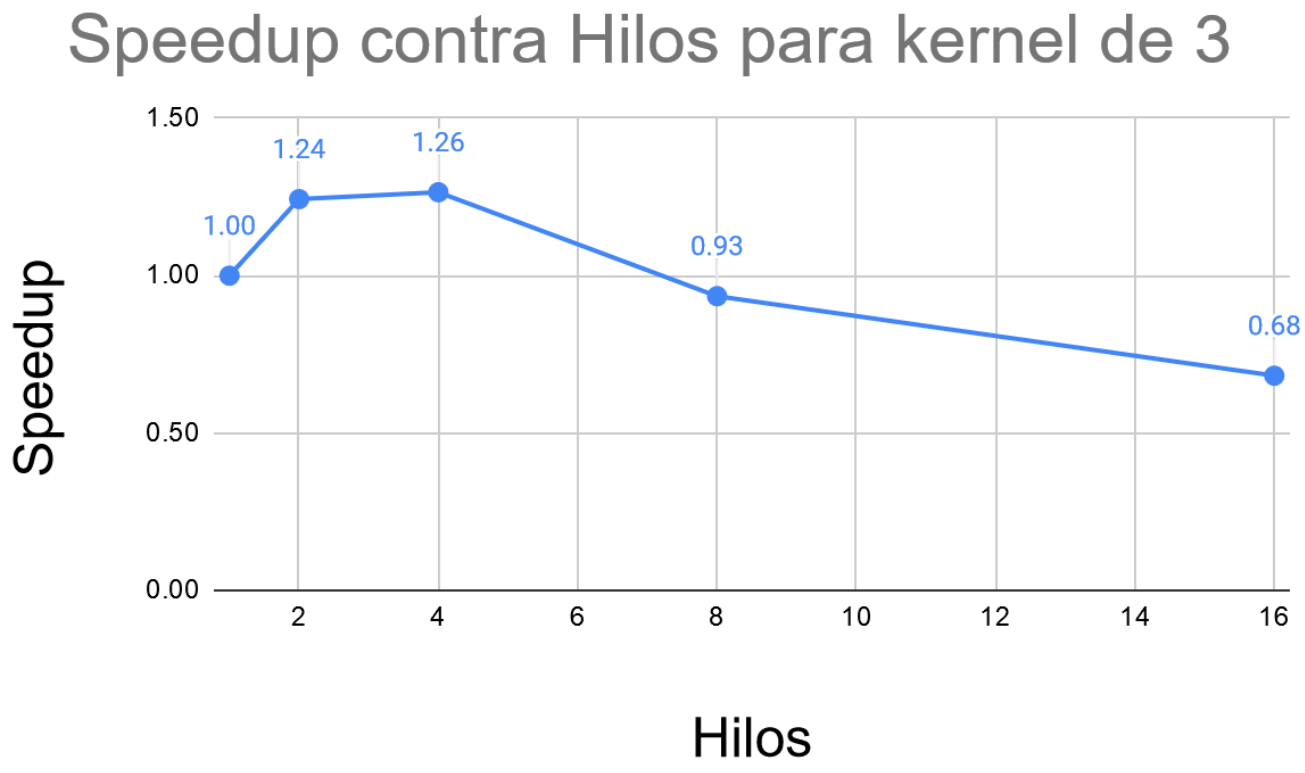
Anexo B
Graficas hilos vs tiempo y Speedup para imagen 1080p





Anexo C
Graficas hilos vs tiempo y Speedup para imagen 4k





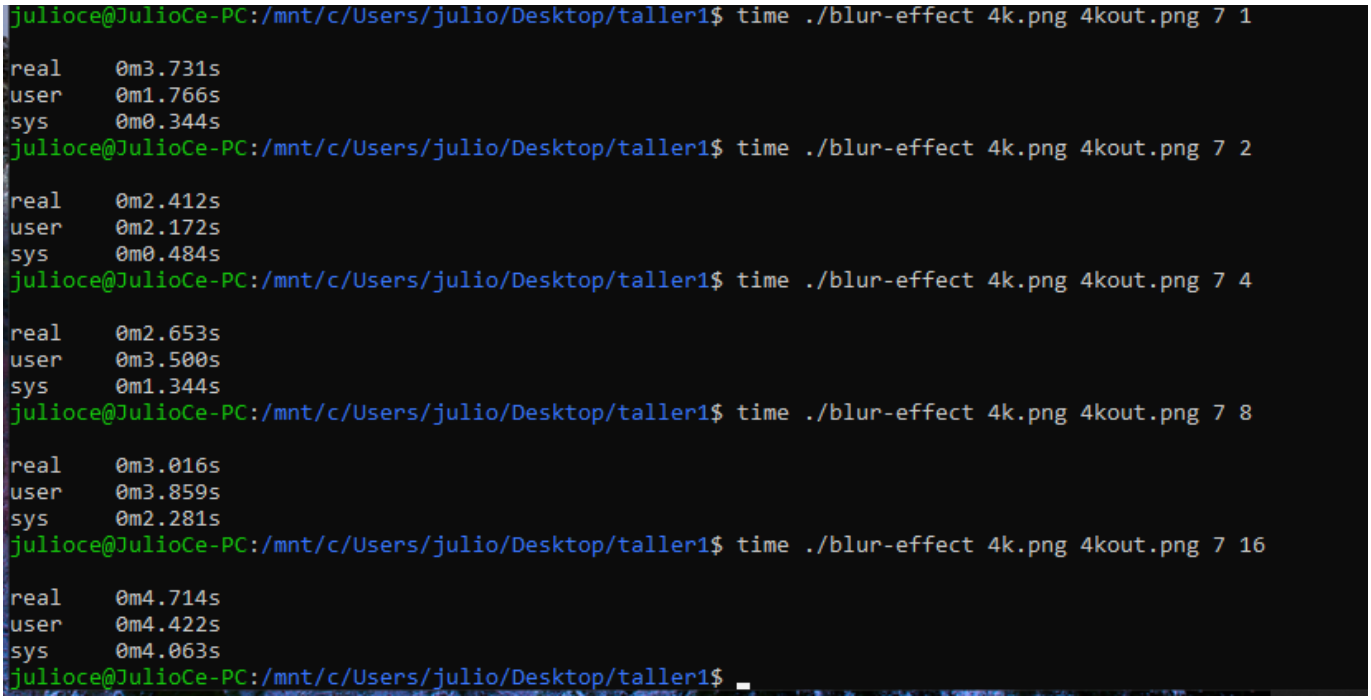
Anexo D
Pruebas adicionales realizadas en otras maquinas

```
julioc2797@instance-2:~$ time ./blur-effect img4k.png BLURimg4k.png 7 1

real    0m1.962s
user    0m1.862s
sys     0m0.100s
julioc2797@instance-2:~$ time ./blur-effect img4k.png BLURimg4k.png 7 2
^[[A
real    0m1.512s
user    0m1.897s
sys     0m0.132s
julioc2797@instance-2:~$ time ./blur-effect img4k.png BLURimg4k.png 7 4

real    0m1.289s
user    0m1.974s
sys     0m0.200s
julioc2797@instance-2:~$ time ./blur-effect img4k.png BLURimg4k.png 7 8

real    0m1.198s
user    0m2.155s
sys     0m0.390s
julioc2797@instance-2:~$ time ./blur-effect img4k.png BLURimg4k.png 7 16

real    0m1.251s
user    0m3.719s
sys     0m0.898s
```

G-cloud

```
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$ time ./blur-effect img4k.png BLURimg4k.png 3 1

real    0m1.355s
user    0m1.078s
sys     0m0.234s
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$ time ./blur-effect img4k.png BLURimg4k.png 3 2

real    0m1.193s
user    0m1.328s
sys     0m0.266s
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$ time ./blur-effect img4k.png BLURimg4k.png 3 4

real    0m1.089s
user    0m1.266s
sys     0m0.625s
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$ time ./blur-effect img4k.png BLURimg4k.png 3 8

real    0m1.368s
user    0m1.516s
sys     0m1.344s
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$ time ./blur-effect img4k.png BLURimg4k.png 3 12

real    0m1.665s
user    0m1.750s
sys     0m2.453s
camilo@DESKTOP-3EP67OA:/mnt/c/Users/Camilo/Documents$
```

R5 3600


julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$ time ./blur-effect 4k.png 4kout.png 7 1
real 0m3.731s
user 0m1.766s
sys 0m0.344s
julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$ time ./blur-effect 4k.png 4kout.png 7 2
real 0m2.412s
user 0m2.172s
sys 0m0.484s
julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$ time ./blur-effect 4k.png 4kout.png 7 4
real 0m2.653s
user 0m3.500s
sys 0m1.344s
julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$ time ./blur-effect 4k.png 4kout.png 7 8
real 0m3.016s
user 0m3.859s
sys 0m2.281s
julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$ time ./blur-effect 4k.png 4kout.png 7 16
real 0m4.714s
user 0m4.422s
sys 0m4.063s
julioce@JulioCe-PC:/mnt/c/Users/julio/Desktop/taller1$

I5-3230M