INFORME PRÁCTICA 3: PROCESAMIENTO DE IMÁGENES CON EFECTO DE DETECCIÓN DE BORDES UTILIZANDO EL OPERADOR SOBEL Y CUDA.

Andres Camilo Correa Romero
correo: anccorrearo@unal.edu.co
Heidy Johanna Alayón Sastoque
correo: halayont@unal.edu.co

RESUMEN: *En este documento se muestra el desarrollo de la práctica 3 donde se compara el rendimiento de un algoritmo para el procesamiento de imágenes con efecto de detección de bordes utilizando el operador sobel, mediante la programación secuencial y luego con CUDA variando la cantidad de hilos y mostrando los resultados obtenidos.para tiempo de respuesta y speedup.*
Las imágenes que se procesaron tienen 720p, 1080p y 4K.

PALABRAS CLAVE: Procesamiento de imágenes, detección de bordes, sobel, cuda.

1 INTRODUCCIÓN

El filtrado de imágenes busca obtener un resultado con determinadas características a partir de una imagen origen, existen diferentes técnicas que se pueden aplicar y que han sido ampliamente documentados. Para el objeto de esta práctica se aplicará la detección de bordes utilizando el operador sobel que se centra en las diferencias de intensidad que se dan pixel a pixel. para obtener los contornos de objetos.

La detección de bordes se refiere al proceso de identificar y localizar discontinuidades nítidas en una imagen. Las discontinuidades son cambios abruptos en la intensidad de los píxeles que caracterizan los límites de un objeto en una escena. El método clásico de detección de bordes implica convolucionar una imagen con un operador a (filtro 2-D), que está construido para ser sensible a un gran gradiente en una imagen mientras devuelve valores de 0 en regiones uniformes.

1.1 OPERADOR SOBEL

El operador Sobel es utilizado en procesamiento de imágenes, especialmente en algoritmos de detección de bordes. Técnicamente es un operador diferencial discreto que calcula una aproximación al gradiente de la función de intensidad de una imagen. Para cada punto de la imagen a procesar, el resultado del operador Sobel es tanto el vector gradiente correspondiente como la norma de este vector.

El resultado muestra cómo de abruptamente o suavemente cambia una imagen en cada punto analizado y, en consecuencia, cuán probable es que este represente un borde en la imagen y, también, la orientación a la que tiende ese borde.

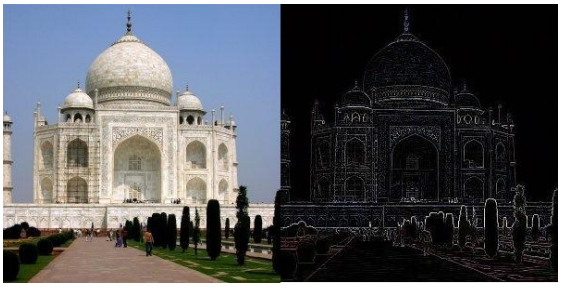


Figura 1. Imagen antes y después de aplicar el operador sobel

1.2 TRATAMIENTO MATEMÁTICO

Cuando se utiliza la detección de bordes Sobel, la imagen se procesa en las direcciones X e Y por separado primero y luego se combina para formar una nueva imagen que representa la suma de los bordes X e Y de la imagen.

Se usa lo que se llama convolución de kernel. Un kernel es una matriz de 3 x 3 que consta de índices ponderados de manera diferente (o simétrica).

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Se pretende implementar este algoritmo en el lenguaje de programación C++, primero de manera secuencial y luego particionar el problema para que pueda ser ejecutado en diferentes bloques e hilos con CUDA, con estas simulaciones se obtiene datos para el tiempo de respuesta en cada ejecución y el speedup. Se comparan estas medidas para cuantificar el desempeño de cada implementación y se muestra de manera gráfica el comportamiento de las mismas,

2 DESARROLLO

Para el operador de sobel, es necesario realizar una conversión de la imagen a escala de grises, posteriormente se realiza un blur gaussiano y finalmente se aplica el kernel de sobel en dirección horizontal y vertical.

Las operaciones que más consumen tiempo son el blur gaussiano y el operador de sobel, debido a esto se decidió paralelizar ambas funciones.
El procedimiento para ambas operaciones es similar, ya que se realiza la convolución de un kernel en cada una, por esto se explicará el desarrollo general.

La imagen es interpretada como un arreglo unidimensional, esto para que sea más sencillo repartir las cargas a cada hilo de manera dinámica. Para evitar el false sharing se realiza una copia de la imagen para cada hilo, de esta manera dos hilos no accederán a los mismos espacios de memoria. A medida que el hilo va realizando la convolución, se va marcando en una matriz auxiliar los píxeles modificados, de esta manera al finalizar se puede copiar a la imagen resultante únicamente la información modificada por el hilo.

El repositorio se encuentra en https://github.com/z4yross/sobel-paralel-kernel.

3 RESULTADOS Y ANÁLISIS

El ejercicio se realizó sobre la siguiente máquina:

Ram	16 GB
Procesador	i7 - 8700k 3.8-4.8GHZ 6 cores 12 threads
Multiprocesadores	10
OS	POP OS basado en ubuntu

Para probar el código se usó la misma imágen con diferentes resoluciones (640p,1280p, 1920p y 6k), los resultados de tiempo de ejecución y SpeedUP se presentan en las siguientes gráficas:

3.1 Gráficas

Figura 2. Imagen original

3.1.1 Imagen de 720p

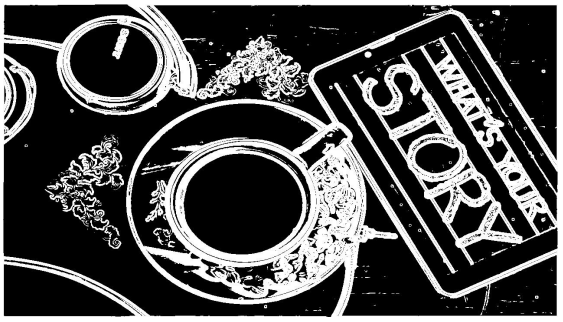
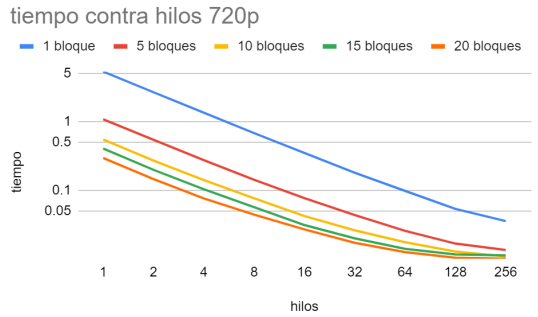


Figura 3: Resultado con imagen de 720p, 20 bloques, 256 hilos,kernel blur:5, Sobel threshold: 2, promedio de 10 experimentos

CUDA 720p					
Bloques	1	5	10	15	20
Hilos	**Tiempo en segundos**				
1	5.279875	1.076703	0.548022	0.405411	0.295791
2	2.650014	0.541678	0.270646	0.19891	0.146141
4	1.339679	0.274414	0.141562	0.104127	0.076631
8	0.677672	0.14242	0.076932	0.057176	0.044638
16	0.349701	0.077333	0.042431	0.031474	0.027236
32	0.1801	0.043945	0.026412	0.020198	0.017418
64	0.09845	0.02596	0.017756	0.014252	0.012728
128	0.053906	0.016964	0.013002	0.01179	0.01059
256	0.035924	0.013686	0.011035	0.011436	0.01021

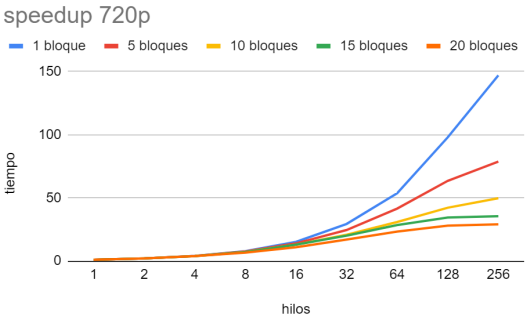
Tabla 1: Tiempo de ejecución promedio (10 intentos) 720p.kernel blur:5, Sobel threshold: 2.



Gráfica 1: Tiempo de ejecución promedio (10 intentos) 720p.kernel blur:5, Sobel threshold: 2.

CUDA 720p					
Bloques	1	5	10	15	20
Hilos	**Speedup**				
1	1	1	1	1	1
2	1.992395135	1.987717795	2.024866431	2.038162988	2.024011058
4	3.941149335	3.923644566	3.871250759	3.893428217	3.859939189
8	7.791195446	7.560054768	7.123459679	7.090579964	6.626439357
16	15.09825537	13.92294363	12.91560416	12.88082227	10.8602952
32	29.31635203	24.50114916	20.74897774	20.07183 88	16.98191526
64	53.63001524	41.47546225	30.86404596	28.44590233	23.23939346
128	97.94596149	63.46987739	42.14905399	34.38600509	27.93116147
256	146.9734718	78.67185445	49.66216584	35.45041973	28.97071499

Tabla 2: Speedup 720p.kernel blur:5, Sobel threshold: 2



Gráfica 2: Speedup 720p.kernel blur:5, Sobel threshold: 2

3.1.1 Imagen de 1080p

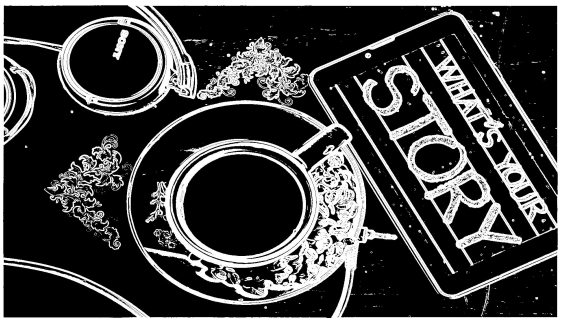
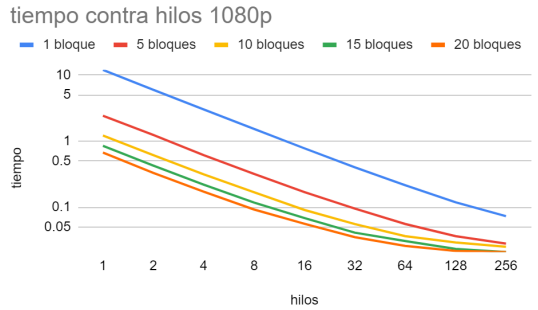


Figura 4. Resultado con imagen de 1080p 20 bloques, 256 hilos,kernel blur:5, Sobel threshold: 2, promedio de 10 experimentos

CUDA 1080p					
Bloques	1	5	10	15	20
Hilos	**Tiempo en segundos**				
1	11.965742	2.416942	1.214563	0.847097	0.670202
2	5.973483	1.239101	0.612233	0.425739	0.330247
4	3.009246	0.61451	0.314295	0.219347	0.17194
8	1.528486	0.318878	0.168687	0.117458	0.092251
16	0.77747	0.168809	0.090894	0.068445	0.056108

32	0.40145	0.095123	0.055582	0.041168	0.035185
64	0.215317	0.055745	0.036431	0.030733	0.026007
128	0.118612	0.036445	0.029239	0.023381	0.02177
256	0.073115	0.028142	0.025194	0.020896	0.020915

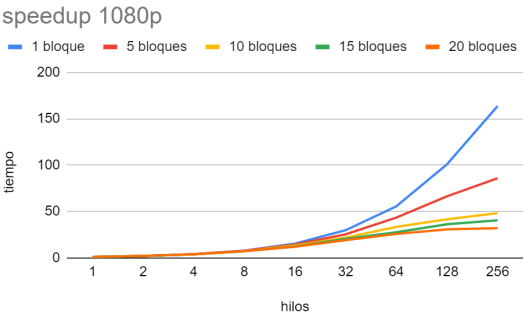
Tabla 3: Tiempo de ejecución promedio (10 intentos) 1080p.kernel blur:5, Sobel threshold: 2.



Gráfica 3: Tiempo de ejecución promedio (10 intentos) vs Hilos 1080p.kernel blur:5, Sobel threshold: 2.

CUDA 1080p					
Bloques	1	5	10	15	20
Hilos	**Speedup**				
1	1	1	1	1	1
2	2.003143225	1.950560931	1.983824786	1.989709658	2.029396179
4	3.976325631	3.933120698	3.864404461	3.861903742	3.897882982
8	7.828493032	7.57951944	7.200098407	7.211914046	7.264983577
16	15.39061572	14.3176134	13.36241116	12.37631675	11.94485635
32	29.80630714	25.40859729	21.85173258	20.57658861	19.04794657
64	55.57267657	43.35710826	33.33872252	27.56310806	25.77006191
128	100.8813779	66.31751955	41.53914293	36.23014413	30.78557648
256	163.656459	85.88380357	48.20842264	40.53871554	32.04408319

Tabla 4: Speedup 1080p.kernel blur:5, Sobel threshold: 2



Gráfica 4: Speedup 1080p.kernel blur:5, Sobel threshold: 2

3.1.1 Imagen de 4k

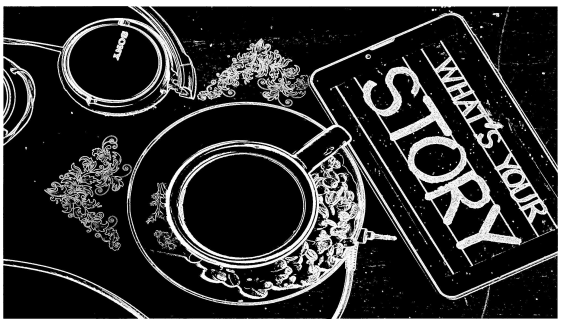
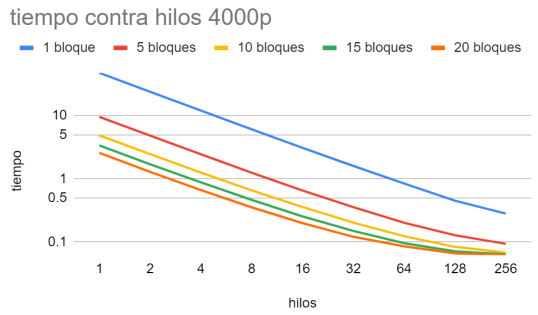


Figura 5. Resultado con imagen de 4k, 20 bloques, 256 hilos,kernel blur:5, Sobel threshold: 2, promedio de 10 experimentos

CUDA 4k					
Bloques	1	5	10	15	20
Hilos	**Tiempo en segundos**				
1	47.552343	9.566288	4.866355	3.388272	2.585055
2	23.89909	4.8244	2.458032	1.701524	1.287632
4	12.04097	2.443364	1.254223	0.878248	0.665084
8	6.076536	1.245735	0.656253	0.464252	0.353931
16	3.092124	0.653275	0.359277	0.254494	0.199256
32	1.600359	0.3572	0.202902	0.149914	0.121193
64	0.844503	0.202029	0.123655	0.095927	0.085526
128	0.451365	0.128214	0.084034	0.071301	0.066395
256	0.282347	0.093739	0.067873	0.06398	0.062078

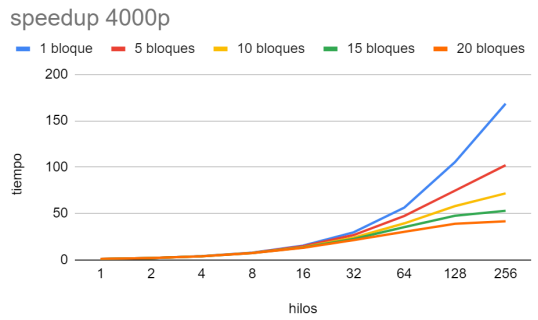
Tabla 5: Tiempo de ejecución promedio (10 intentos) 4k. kernel blur:5, Sobel threshold: 2.



Gráfica 5: Tiempo de ejecución promedio (10 intentos) vs Hilos 4k. kernel blur:5, Sobel threshold: 2.

CUDA 4k					
Bloques	1	5	10	15	20
Hilos	**Speedup**				
1	1	1	1	1	1
2	1.989713541	1.982896941	1.979776911	1.99131602	2.007603881
4	3.949211982	3.915211978	3.879975889	3.857989998	3.886809786
8	7.82556756	7.679231939	7.415364196	7.298346588	7.303838884
16	15.37853689	14.64358501	13.54485536	13.31375985	12.97353656
32	29.7135474	26.78132139	23.98377049	22.60143816	21.33006857
64	56.30808061	47.35106346	39.35429218	35.32135895	30.22537006
128	105.3523047	74.61188326	57.90935812	47.52067993	38.93448302
256	168.4180919	102.0523795	71.69795058	52.95829947	41.6420471

Tabla 6: Speedup 4k.kernel blur:5, Sobel threshold: 2



Gráfica 6: Speedup 4k. kernel blur:5, Sobel threshold: 2

4 CONCLUSIONES

- Para el óptimo funcionamiento del operador sobel es preferible que la imagen esté en escala de grises ya que así no se genera tanto ruido y es más eficiente al detectar los bordes.
- Es necesario utilizar blur para mejorar la nitidez de la imagen final aunque el sobel es capaz de detectar los bordes por si mismo.
- Se observa una reducción sustancial de tiempo, en comparación con el uso de posix y openmp, con el uso de entre 10 y 20 bloques y entre 64 y 256 hilos
- Según las gráficas el número óptimo de bloques sería 15 y 128 hilos ya que a partir de ahí el tiempo de ejecución se va estabilizando.
- El resultado de la ejecución de la práctica fue consistente con la teoría
- El uso de hilos incrementa el rendimiento del procesador
- Es importante tener control de los hilos para que el resultado sea el esperado y no existan conflictos entre ellos.

5 REFERENCIAS

1 Preparación de informes en formato IEEE, http://utap.edu.co/ccys/wp-content/uploads/2018/Eventos/IEEE/Paper_IEEE_Oficial_Espa%C3%B1ol.pdf.
2 Derivados Sobel, https://programmerclick.com/article/61448242/.